Python Básico

Osenias Oliveira

Que sou eu?

- Mais de 15 anos de experiência com Desenvolvimento de Software, Liderança de Projetos, Levantamento de Requisitos, Administração e Tunning de Banco de Dados.
- Analista de Sistemas Master na **FPF Tech** desde março de **2014**.
- Professor do Curso Técnico na Fucapi entre **2011** e **2016**.
- Certificação Scrum Master.
- Certificação Product Owner.
- Pós Graduado em Projeto e Administração de Banco de Dados pela **Uninorte**.
- Graduado em Desenvolvimento de Software pela **Uninorte**.

E o que nós vamos aprender?

- Definição
- Ranking de Linguagens de Programação
- Porque usar Python?
- Quem usa Python?
- Aplicabilidade da Linguagem Python
- Instalação e configuração do ambiente
- Gerenciador de pacotes **PIP**
- Virtual Environment
- Tipos básicos
- Operadores
- Listas, Tuplas e Dicionários
- Estruturas de controle
- Estruturas de repetição
- Funções
- Funções parciais

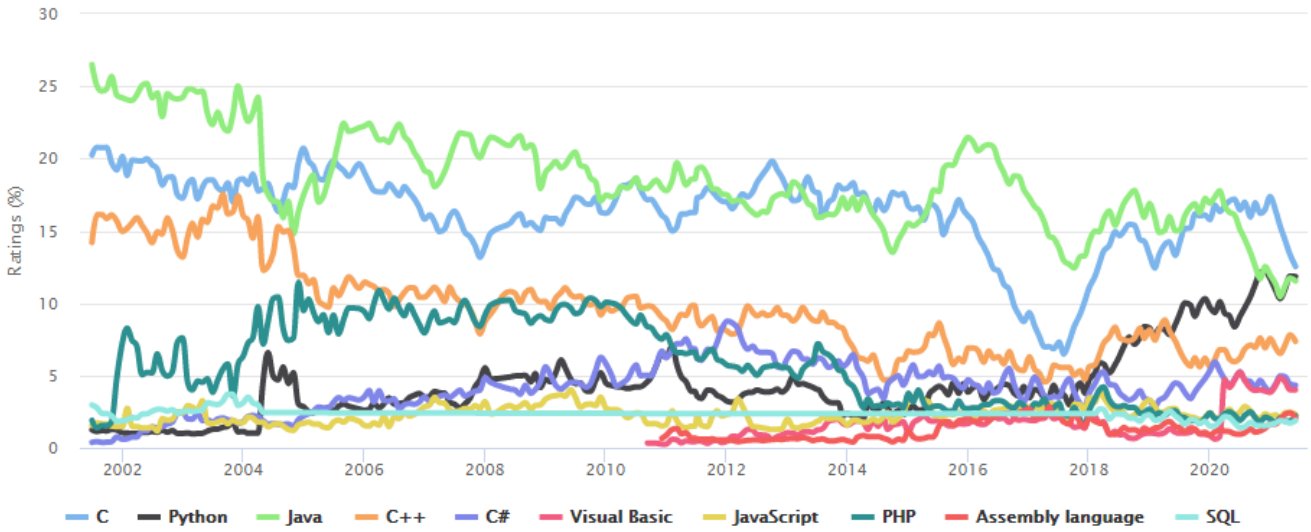
E o que nós vamos aprender?

- Funções de string
- Funções built-in
- Compreensão de listas
- Iterators
- Arquivos
- Orientação a objetos
- Classes, atributos e métodos
- Herança
- Módulos
- Importações
- Exceções
- Testes unitários
- Outros pacotes

Definição

- Python é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por **Guido van Rossum** em 1991.
- É reconhecida por ser muito versátil, com ênfase na legibilidade do código, e por ter uma abordagem que permite aos programadores desenvolverem algoritmos melhor estruturados. Ela é orientada por uma lista de 19 princípios chamada de **"The Zen of Python"**.
  - Bonito é melhor do que feio
  - Explícito é melhor do que subentendido
  - Simples é melhor do que complexo
  - Legibilidade é importante.

Ranking de Linguagens de Programação



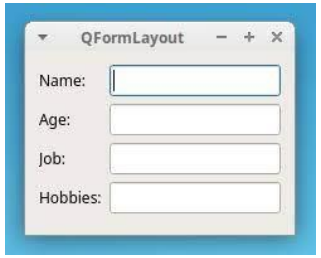
Porque usar Python?

- Facilidade
- Aumento de produtividade
- Comunidade ampla e popularidade
- Utilização Versátil
- Inteligência Artificial
- Programação Web
- Gerenciamento de Big Data
- Computação Gráfica
- Automação
- Mercado Amplo
- Bons Salários

Aplicabilidade da Linguagem Python

- Aplicações Web

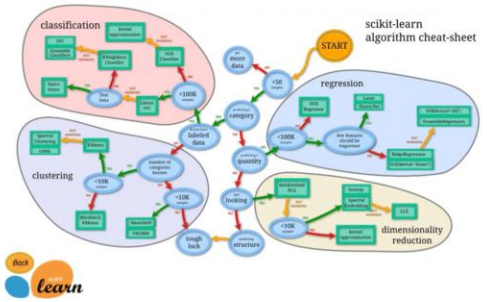
- Aplicações Desktop



Aplicabilidade da Linguagem Python

- Embarcado

- Análise de Dados / Aprendizado de Máquina

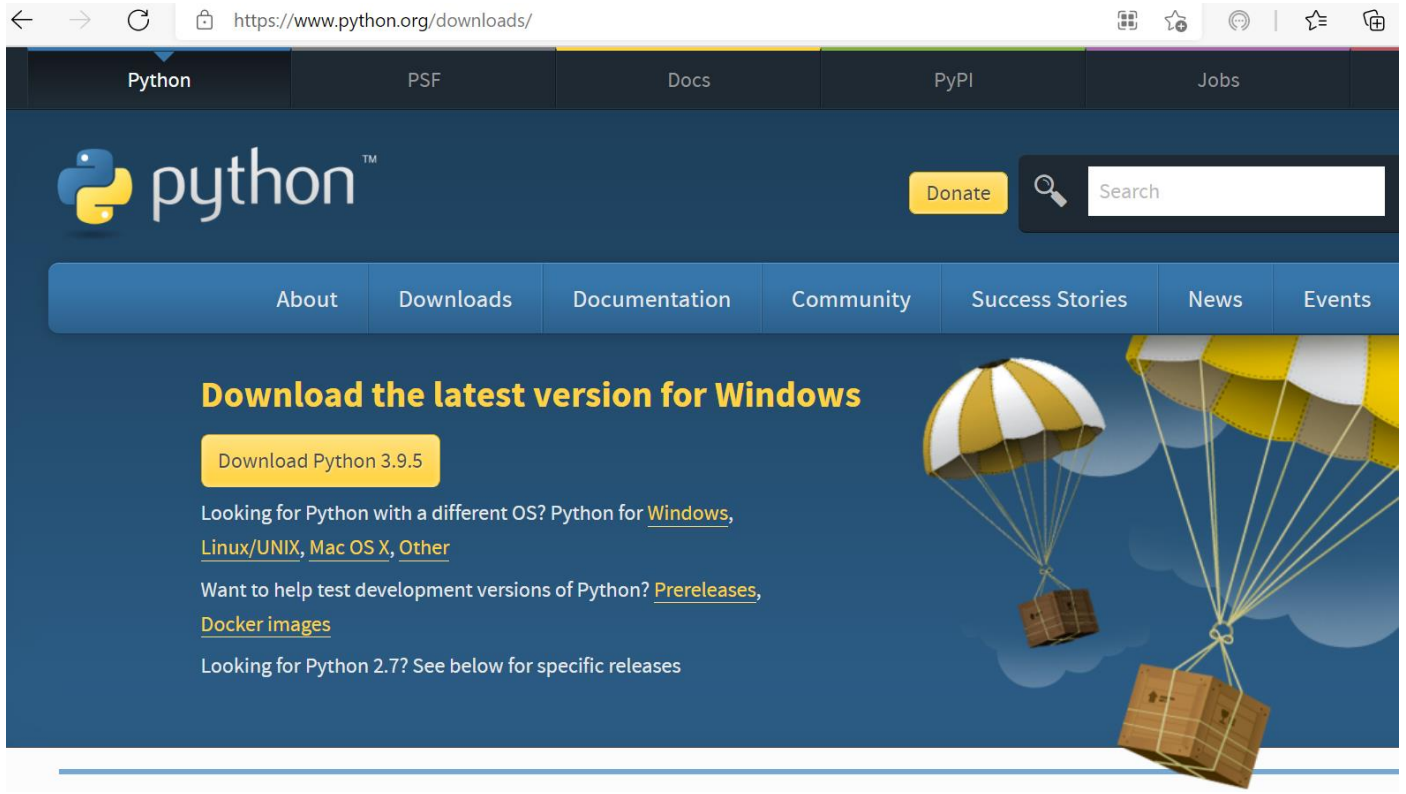


Quem usa Python?

- **Instagram:** Utiliza Django como backend
- **Google:** Grande parte do algoritmo de busca é escrito em python
- **Spotify:** O Aplicativo é construído em python
- **Netflix:** Utiliza diversas bibliotecas em python em serviços
- **Uber:** Boa parte do aplicativo é feito com python
- **Dropbox:** contratou o criador da linguagem Guido van Rossum
- **Pinterest:** utiliza python e Django

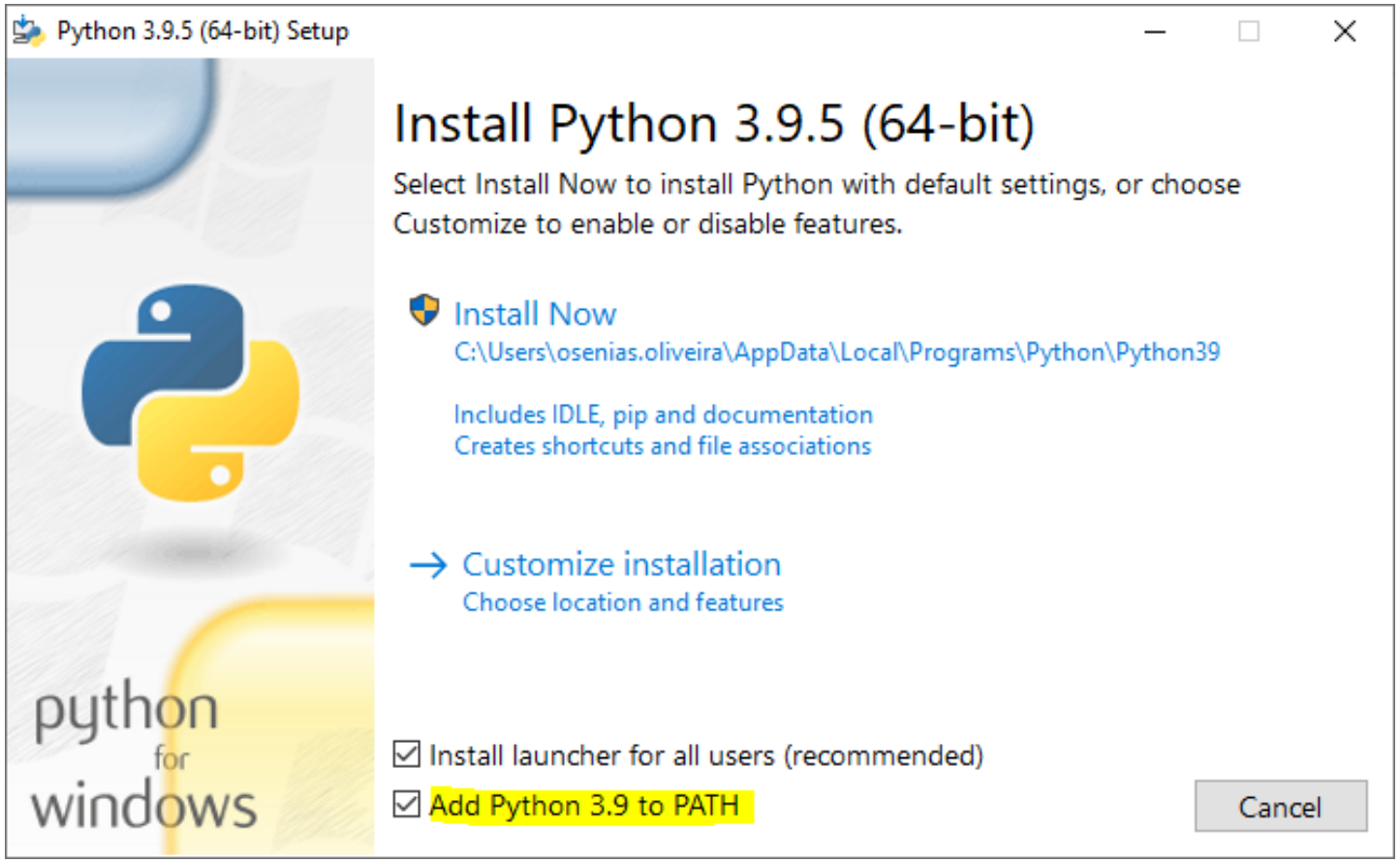
Instalação e Configuração do Ambiente

- Download do interpretador



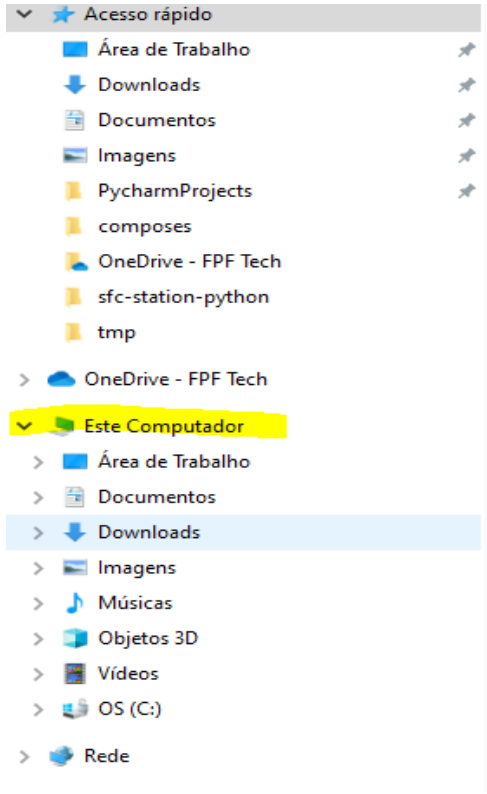
Instalação e Configuração do Ambiente

- Instalação do Python



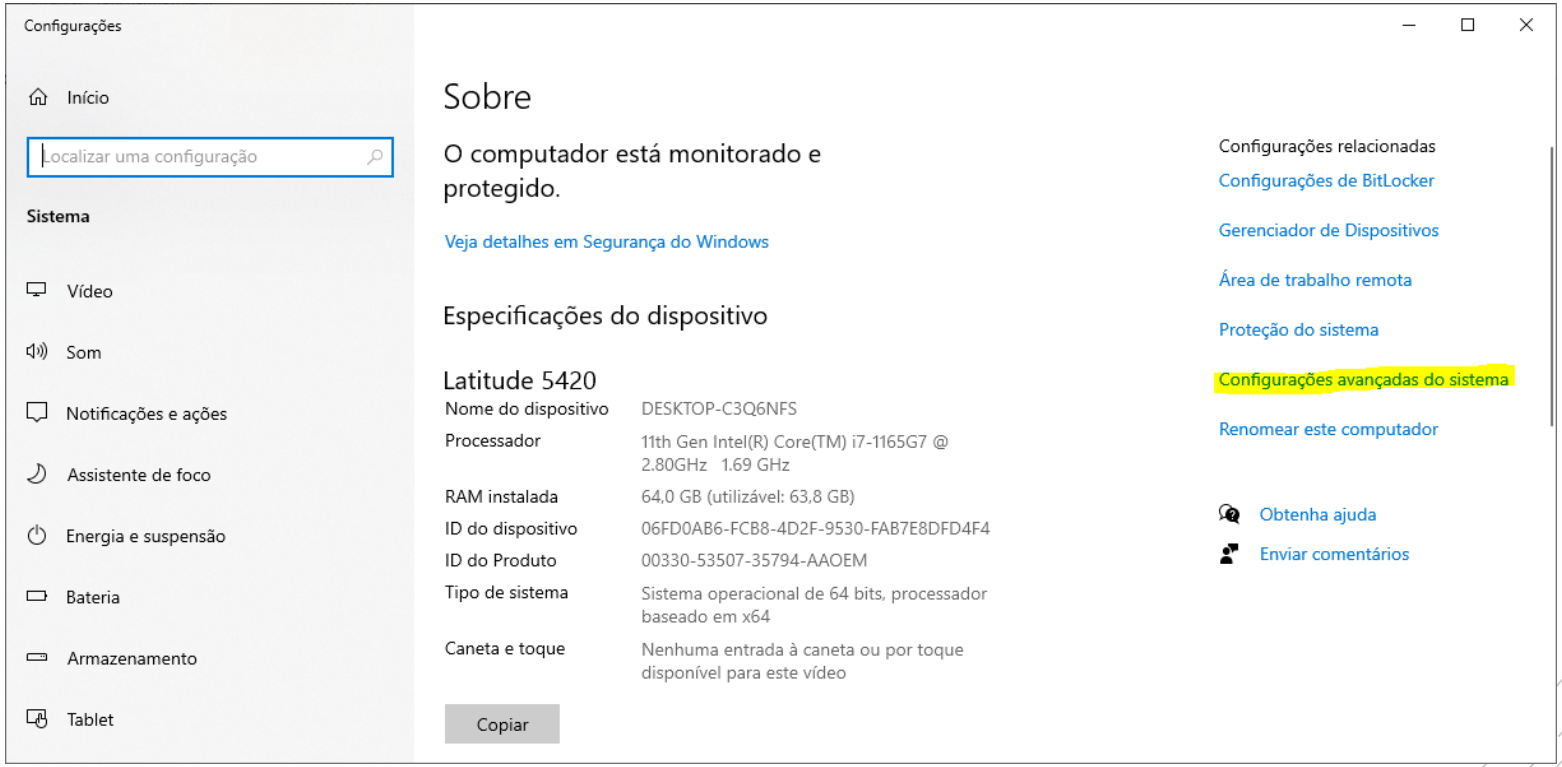
Instalação e Configuração do Ambiente

- Esqueci de marcar a opção para adicionar a variável de ambiente no **path** ☹
- Abra o Explorer do Windows clique o botão direito na opção **Este Computador**



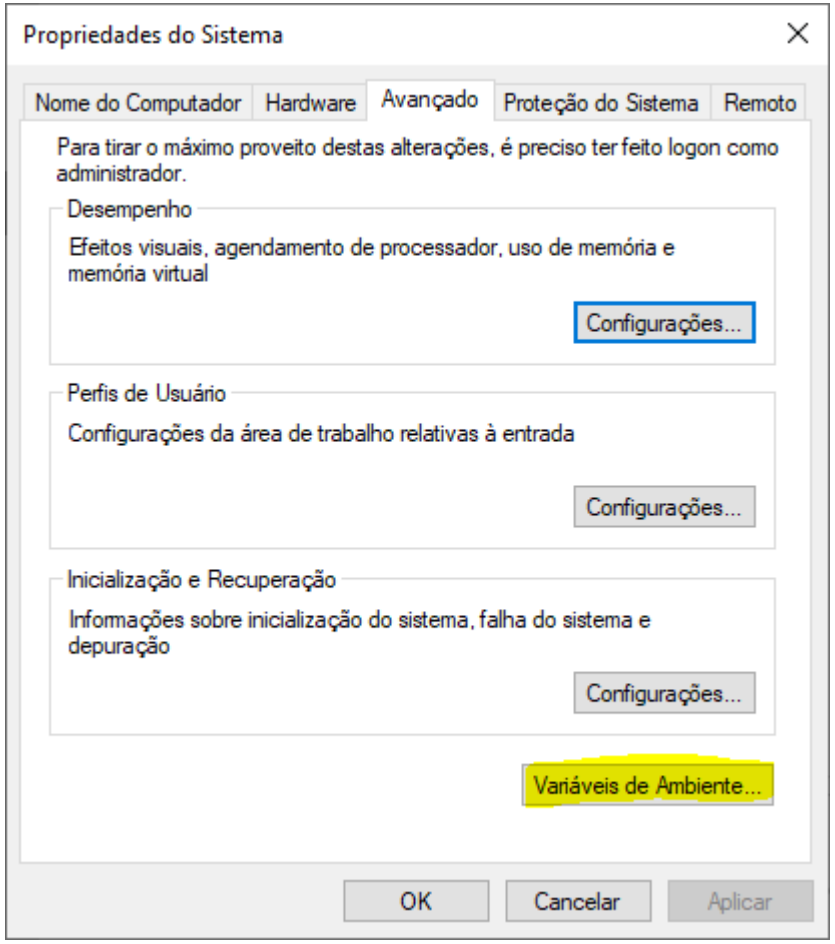
Instalação e Configuração do Ambiente

- Clique na opção **Configurações avançadas do sistema**



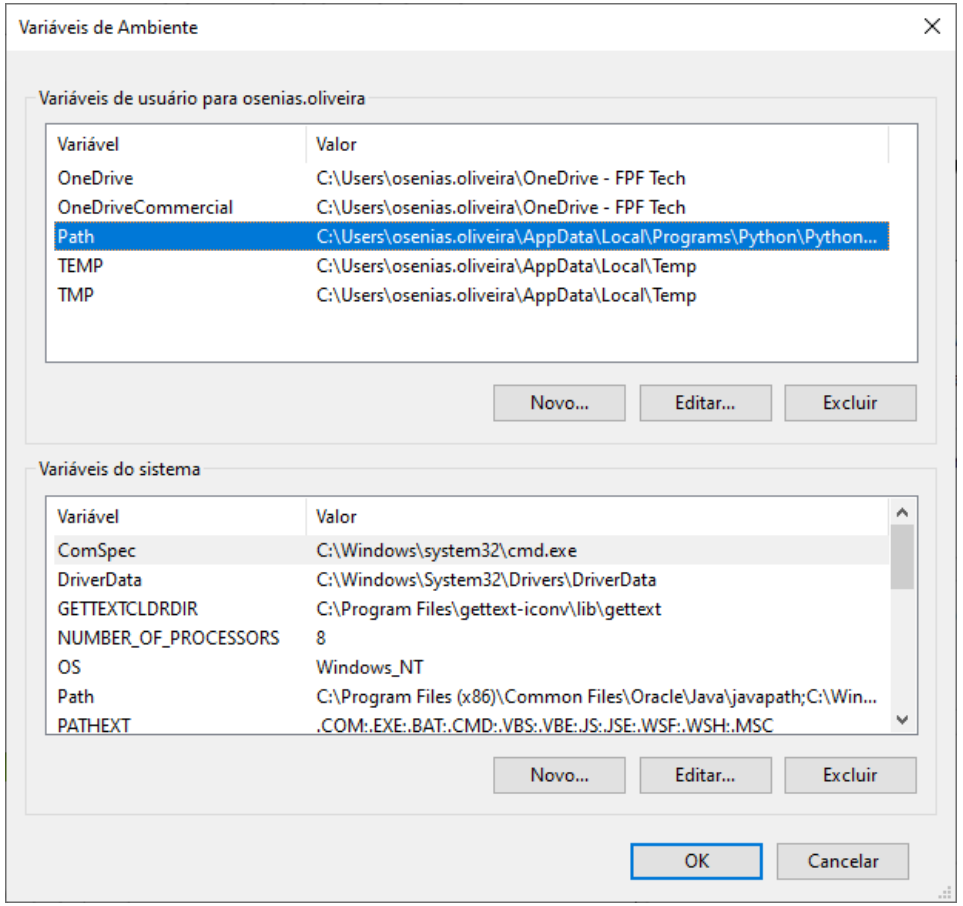
Instalação e Configuração do Ambiente

- Clique na opção **Variáveis de Ambiente**



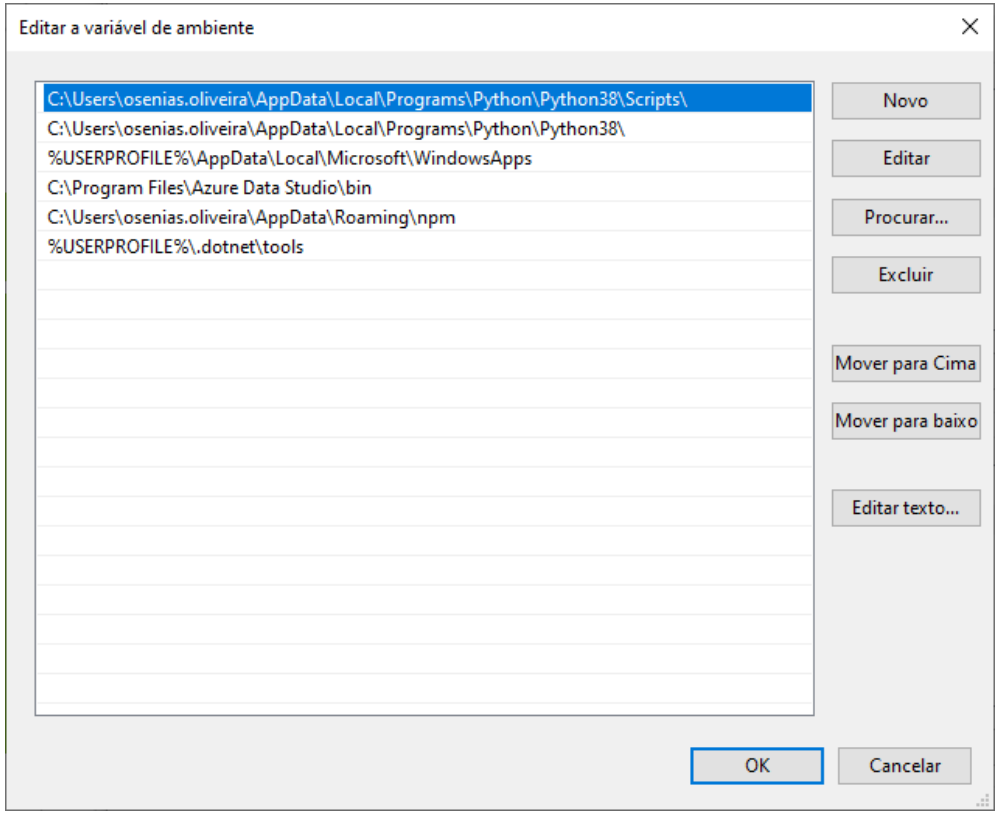
Instalação e Configuração do Ambiente

- Edite o a variável **path**



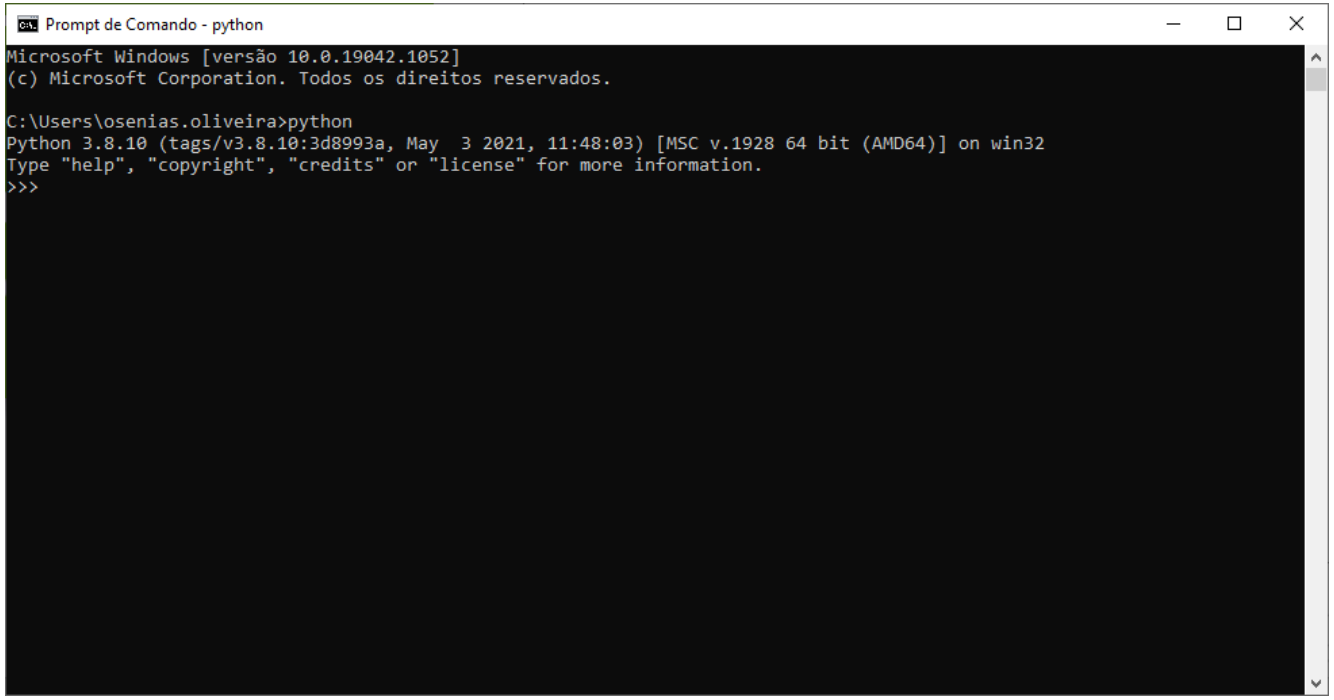
Instalação e Configuração do Ambiente

- Adicione os caminhos da instalação do python



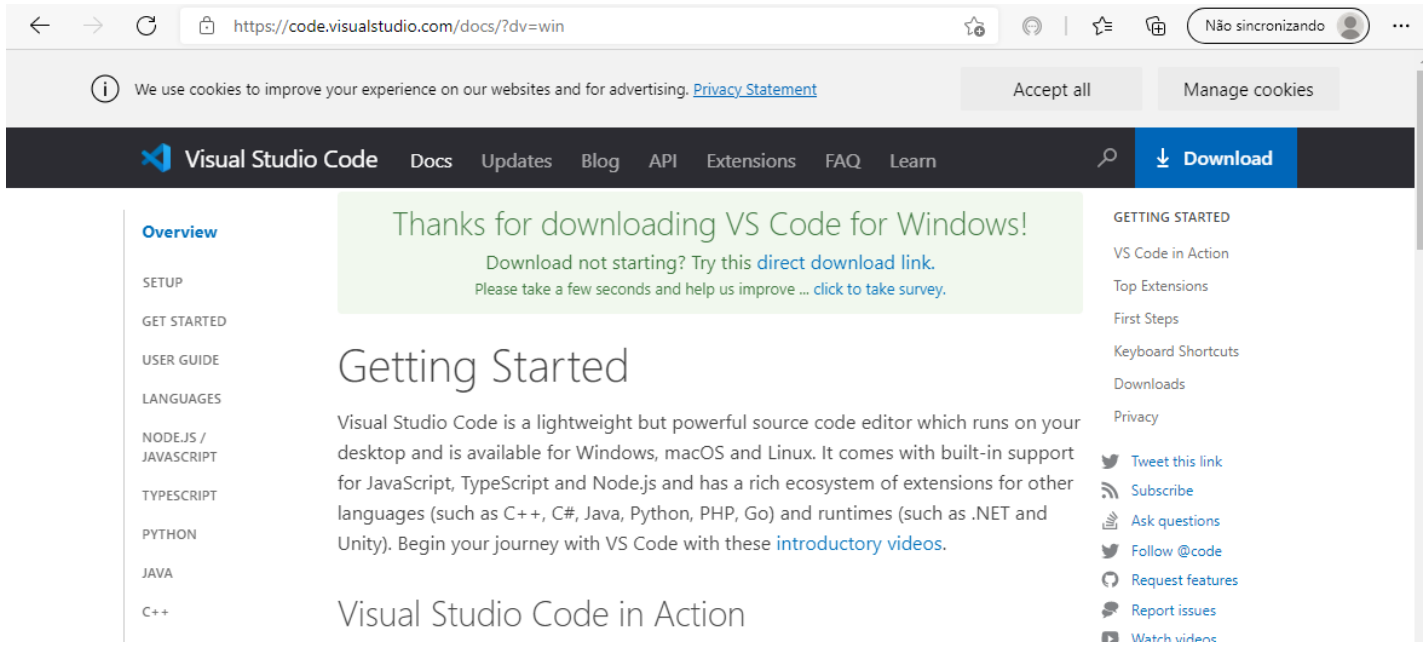
Instalação e Configuração do Ambiente

- Abra o **prompt de comando** para confirmar a instalação e configuração do python



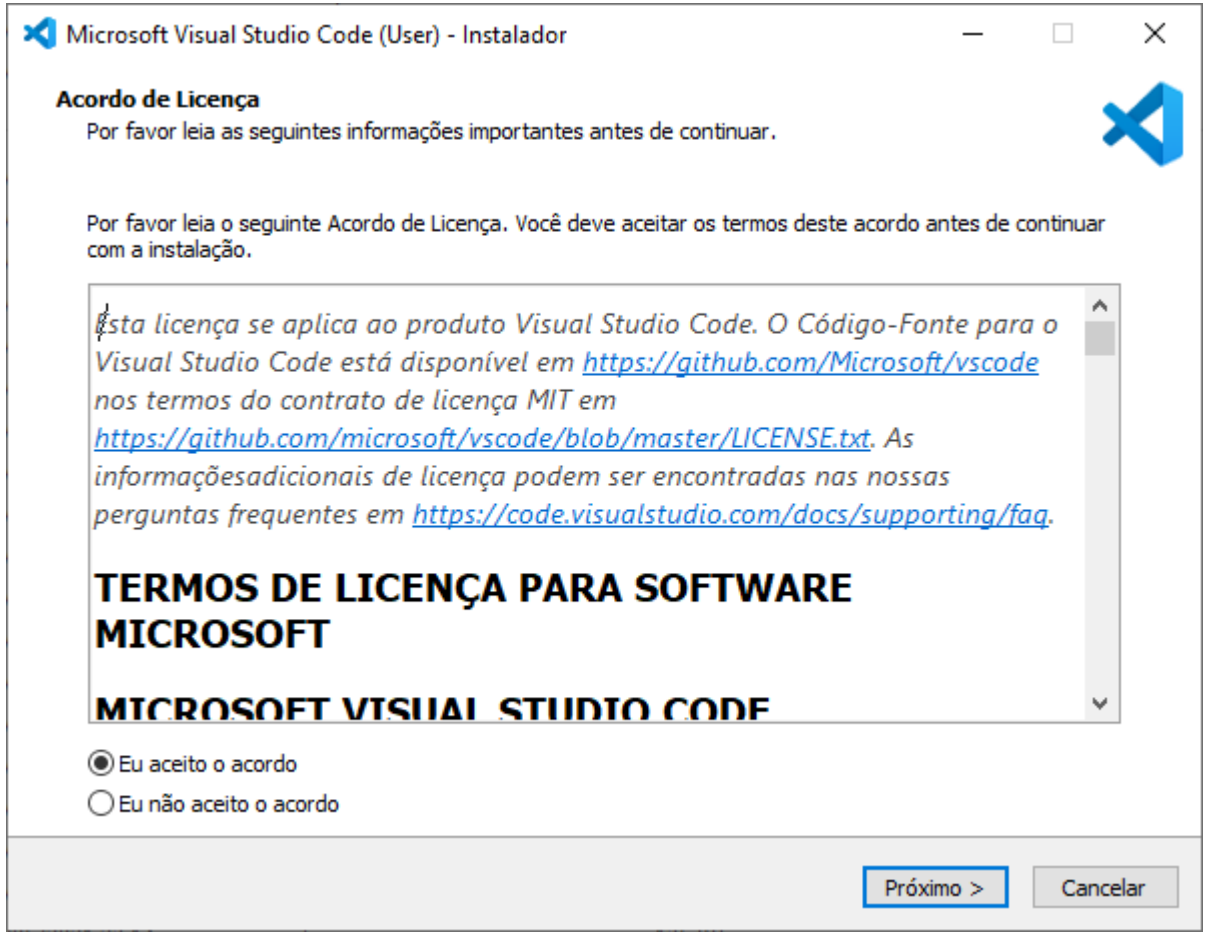
Instalação e Configuração do Ambiente

- **Download VS Code**



Instalação e Configuração do Ambiente

- Instalação do VS Code



Instalação e Configuração do Ambiente

- Instalando a extensão do python

Gerenciador de pacotes PIP

- **pip** é um sistema de gerenciamento de pacotes padrão de facto usado para instalar e gerenciar pacotes de software escritos em Python.
- Muitos pacotes podem ser encontrados na fonte padrão para pacotes e suas dependências - Python Package Index.
- A maioria das distribuições do Python vem com o pip pré-instalado.

Gerenciador de pacotes PIP

- Vamos instalar nosso primeiro pacote.

```
C:\Users\osenias.oliveira>pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.4.7-py2.py3-none-any.whl (7.2 MB)
     |████████████████████████████████| 7.2 MB 1.7 MB/s
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.2-py2.py3-none-any.whl (338 kB)
     |████████████████████████████████| 338 kB ...
Collecting appdirs<2,>=1.4.3
  Using cached appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Collecting six<2,>=1.9.0
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting filelock<4,>=3.0.0
  Downloading filelock-3.0.12-py3-none-any.whl (7.6 kB)
Installing collected packages: six, filelock, distlib, appdirs, virtualenv
Successfully installed appdirs-1.4.4 distlib-0.3.2 filelock-3.0.12 six-1.16.0 virtualenv-20.4.7
WARNING: You are using pip version 21.1.1; however, version 21.1.2 is available.
You should consider upgrading via the 'c:\users\osenias.oliveira\appdata\local\programs\python\python38\python.exe -m
p install --upgrade pip' command.
```

Virtual Environment

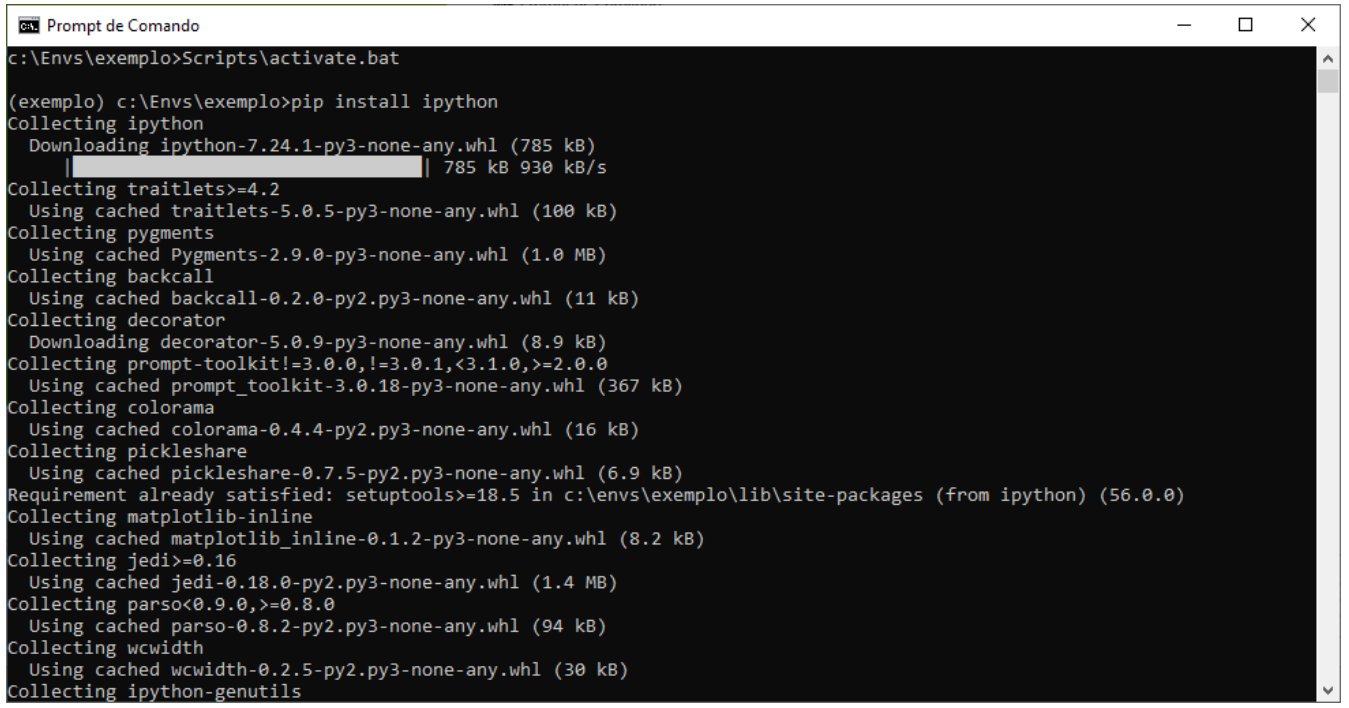
- Ativando a **env**

Prompt de Comando

```
C:\Users\osenias.oliveira>python -m venv c:\Envs\exemplo

C:\Users\osenias.oliveira>cd c:\Envs\exemplo

c:\Envs\exemplo>Scripts\activate.bat

(exemplo) c:\Envs\exemplo>
```

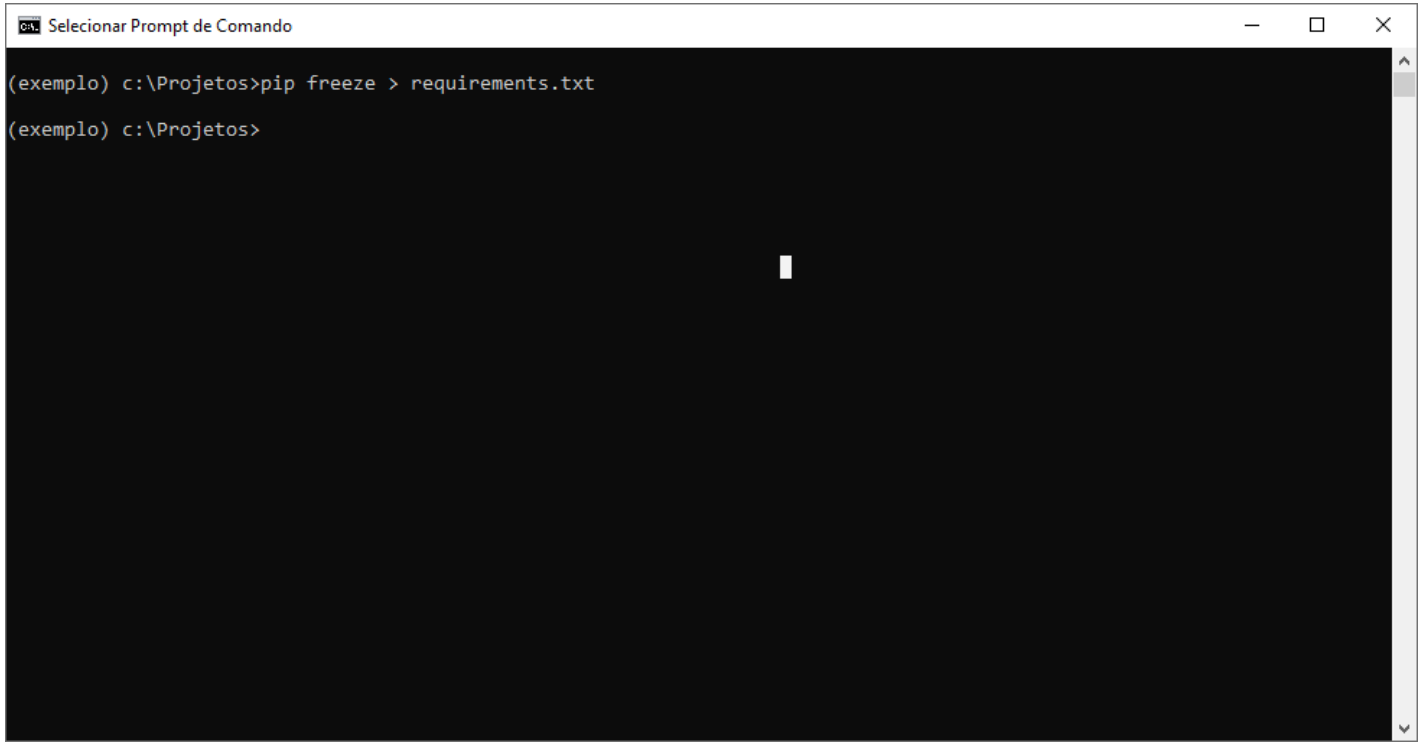
Virtual Environment

- Instalação de um pacote



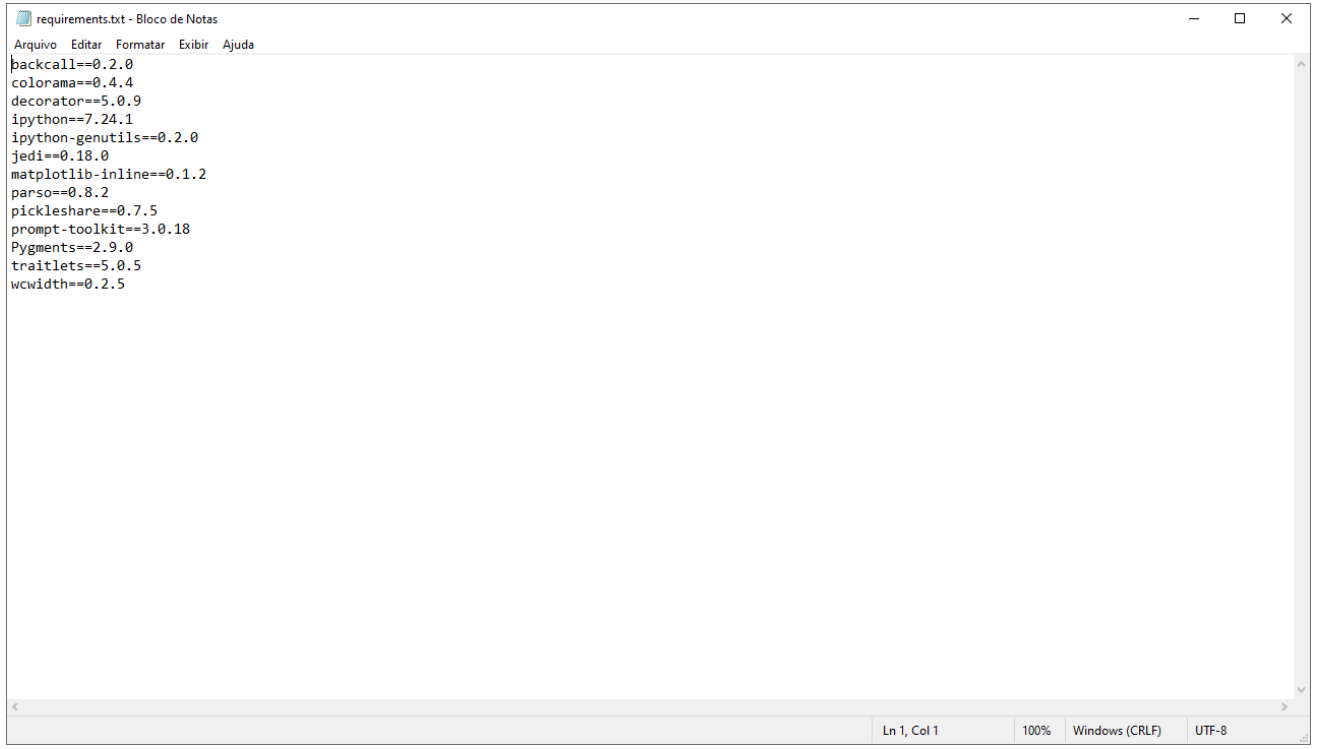
Virtual Environment

- Criando um arquivo para distribuir seus pacotes



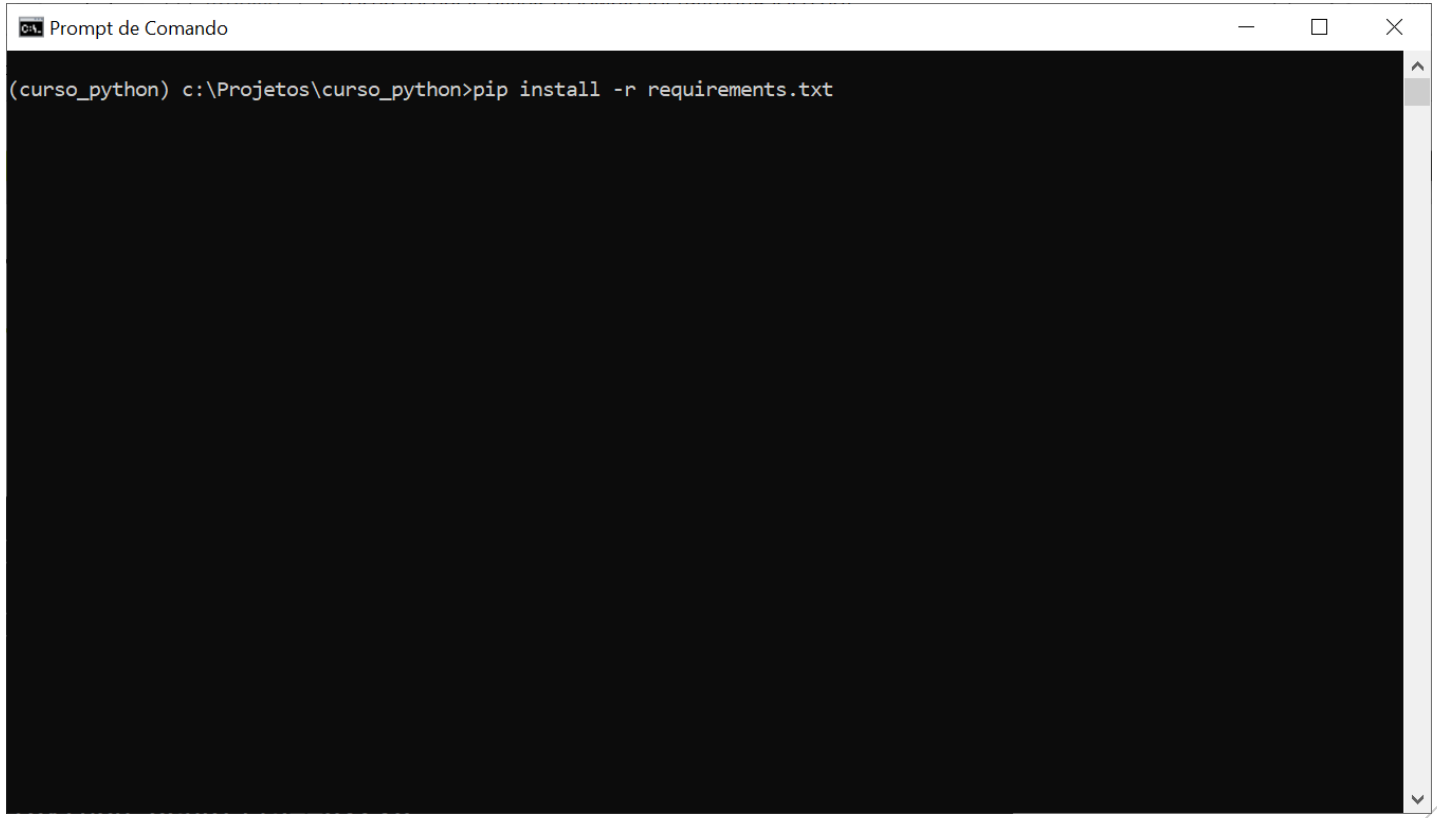
Virtual Environment

- Criando um arquivo para distribuir seus pacotes



Virtual Environment

- Para instalar todos os itens do **requirements.txt** em uma env vazia



Tipos Básicos

- Números inteiros

```
(exemplo) c:\Projetos>ipython
Python 3.8.10 (tags/v3.8.10:3d8993a, May  3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.24.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: idade = 10

In [2]: print(idade)
10

In [3]: peso = 80

In [4]: 80
Out[4]: 80

In [5]: soma = idade + peso

In [6]: soma
Out[6]: 90

In [7]: type(soma)
Out[7]: int

In [8]: type(idade)
Out[8]: int
```

```
In [16]: idade : int = 33

In [17]: idade
Out[17]: 33

In [18]:
```

Tipos Básicos

- Números com pontos flutuante

```
In [14]: quantidade = 100.80

In [15]: altura = 1.62

In [16]: type(quantidade)
Out[16]: float

In [17]: type(altura)
Out[17]: float

In [18]:
```

```
In [20]: quantidade : float = 10.50

In [21]: quantidade
Out[21]: 10.5

In [22]:
```

Tipos Básicos

- Strings

```
In [19]: empresa = "FPF Tech"

In [20]: nome_curso = 'Curso Básico de Python'

In [21]: resultado = empresa + ' - ' + nome_curso

In [22]: resultado
Out[22]: 'FPF Tech - Curso Básico de Python'

In [23]: type(resultado)
Out[23]: str

In [24]: interpolacao = f'{empresa} - {nome_curso}'

In [25]: interpolacao
Out[25]: 'FPF Tech - Curso Básico de Python'

In [26]: outro_tipo_interpolacao = '{} - {}'.format(empresa, nome_curso)

In [27]: outro_tipo_interpolacao
Out[27]: 'FPF Tech - Curso Básico de Python'

In [28]:
```

```
In [23]: nome : str = 'Osenias Oliveira'

In [24]: nome
Out[24]: 'Osenias Oliveira'

In [25]:
```

Tipos Básicos

- Boleano

```
In [29]: verdadeiro = True

In [30]: falso = False

In [31]: verdadeiro
Out[31]: True

In [32]: falso
Out[32]: False

In [33]: type(verdadeiro)
Out[33]: bool

In [34]: verdadeiro is True
Out[34]: True

In [35]: falso is False
Out[35]: True

In [36]:
```

```
In [26]: verdadeiro : bool = True

In [27]: verdadeiro
Out[27]: True

In [28]:
```

Operadores

- Operadores aritméticos

```
In [37]: soma = 10 + 10

In [38]: subtracao = 10 - 10

In [39]: multiplicacao = 10 * 10

In [40]: divisao = 10 / 10

In [41]: soma
Out[41]: 20

In [42]: subtracao
Out[42]: 0

In [43]: divisao
Out[43]: 1.0

In [44]: multiplicacao
Out[44]: 100

In [45]:
```

Operadores

- Operadores relacionais

```
In [48]: 10 == 10
Out[48]: True

In [49]: 10 >= 5
Out[49]: True

In [50]: 10 <= 5
Out[50]: False

In [51]: 10 < 5
Out[51]: False

In [52]: 10 > 5
Out[52]: True

In [53]:
```

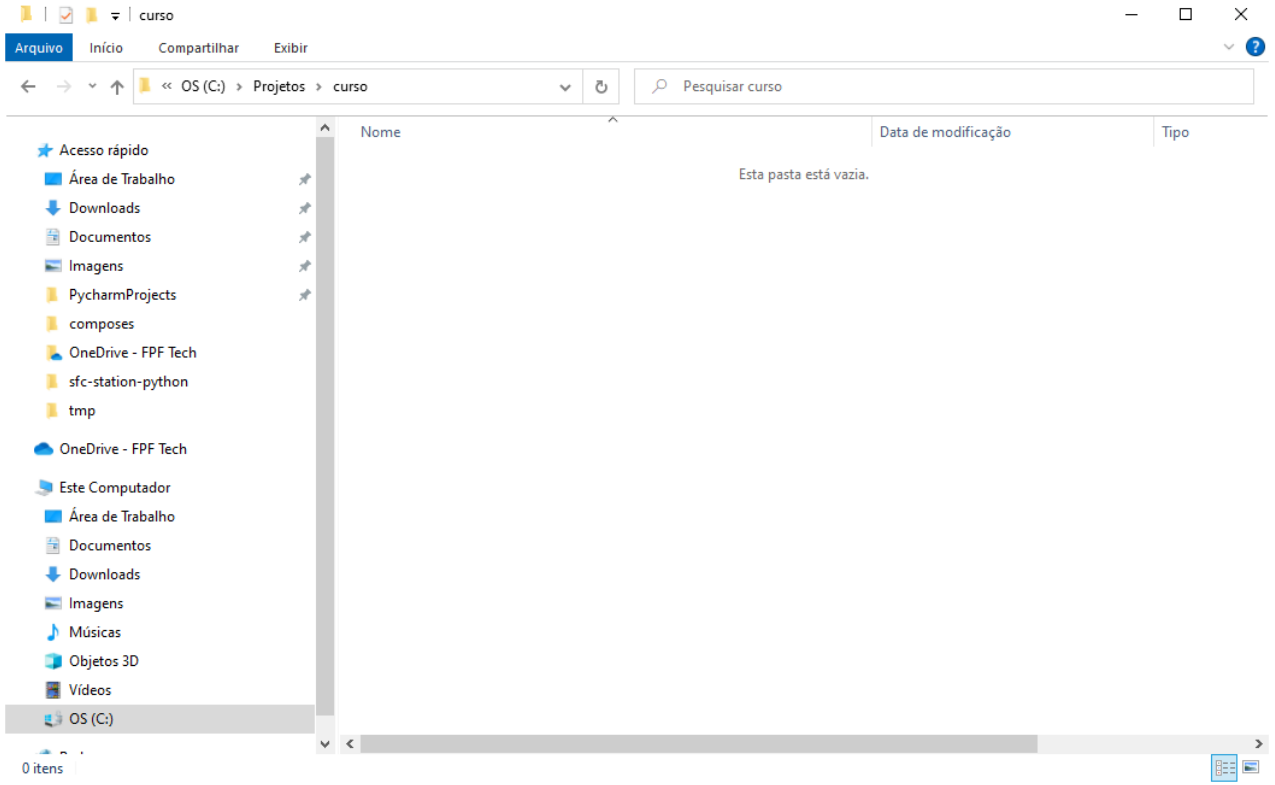
Operadores

- Operadores lógicos

```
In [9]: 10 == 5 or 5 == 5
Out[9]: True

In [10]: 10 == 5 and 5 == 5
Out[10]: False

In [11]: 'o' in 'osenias'
Out[11]: True

In [12]: expression = 10 == 5 or 5 == 5

In [13]: expression
Out[13]: True

In [14]: not expression
Out[14]: False
```

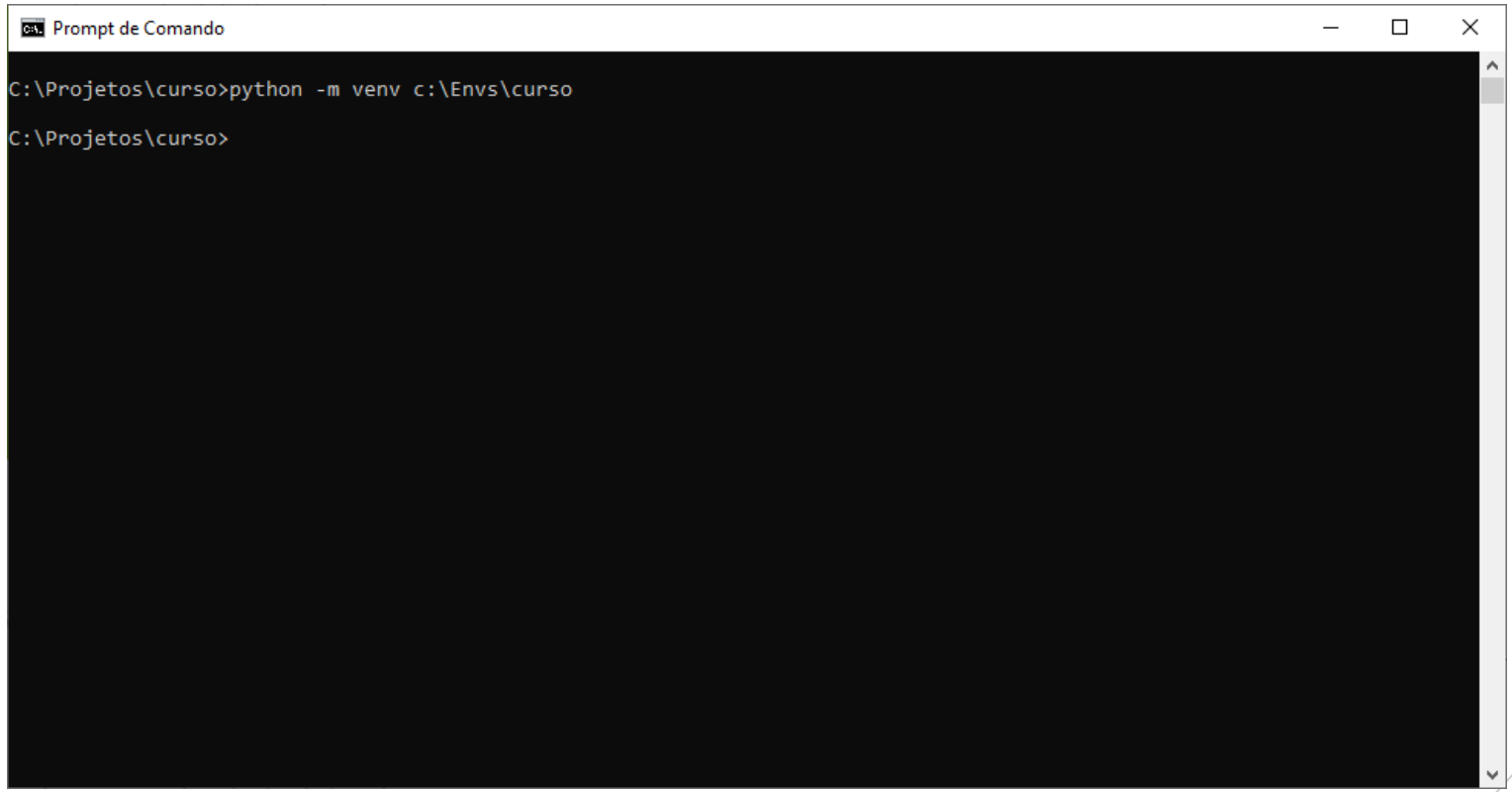
Listas

- Antes de começarmos a falar de lista, vamos criar um novo projeto para que possamos trabalhar dentro do **VS Code**.



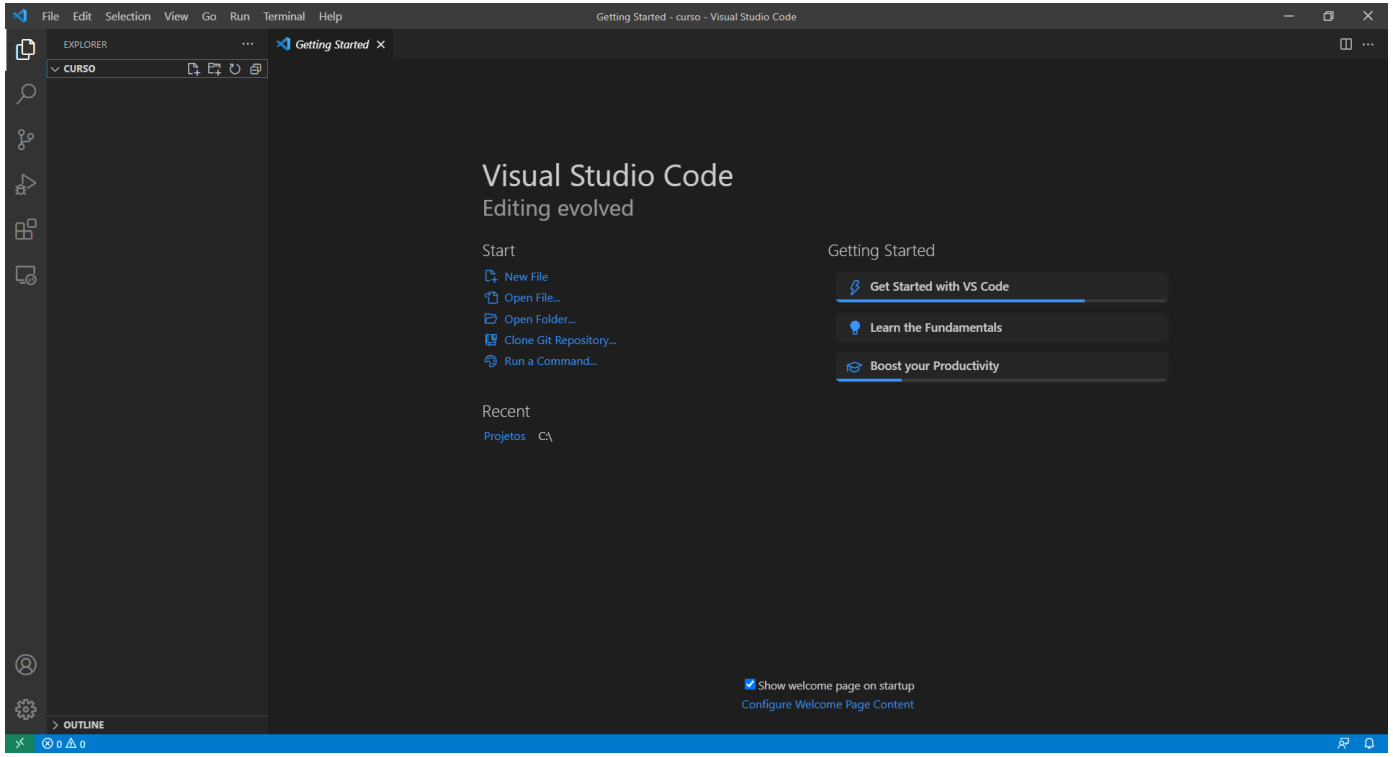
Listas

- Vamos também criar uma **env** específica para o projeto.



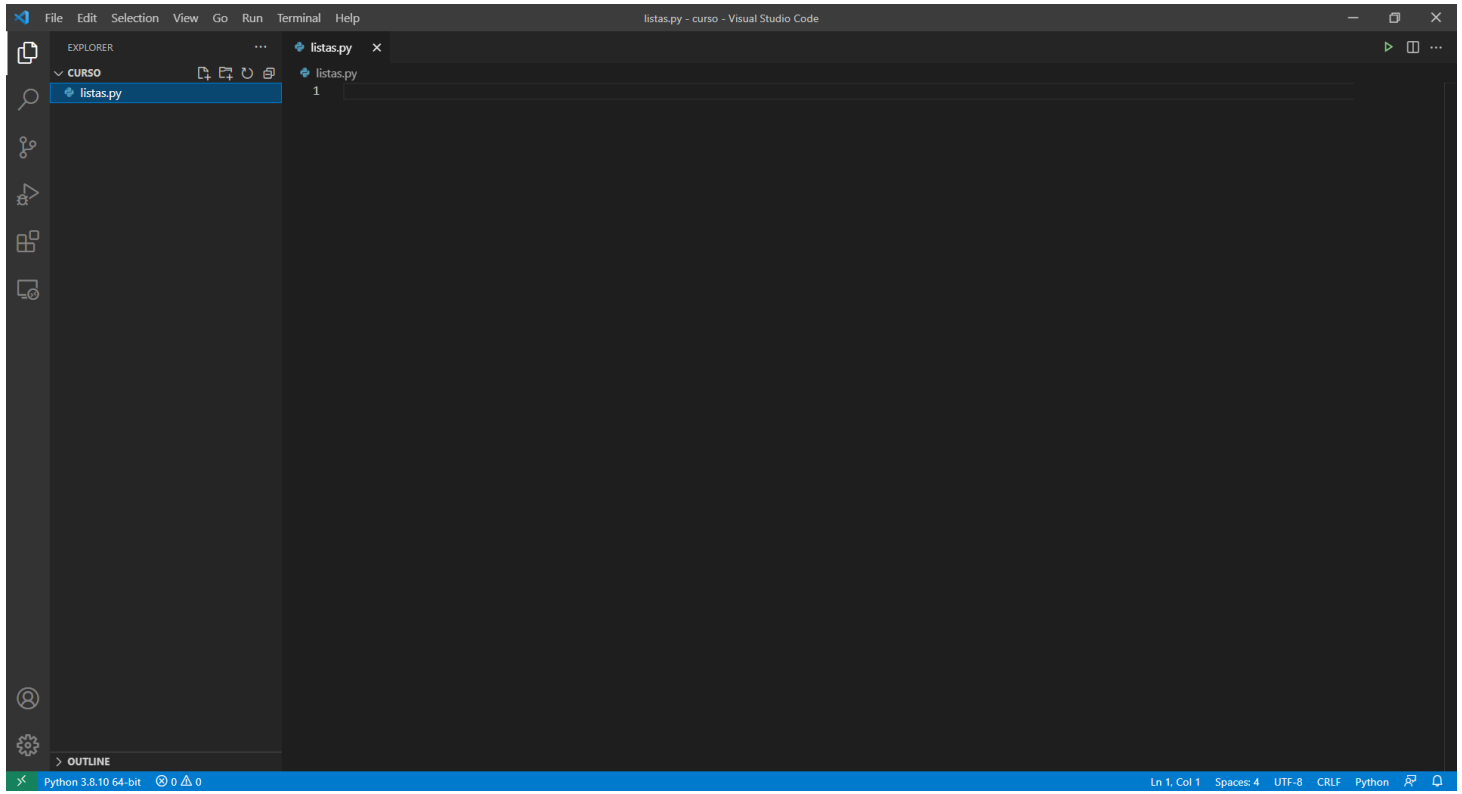
Listas

- Agora abra o **VS Code** e selecione a pasta do projeto



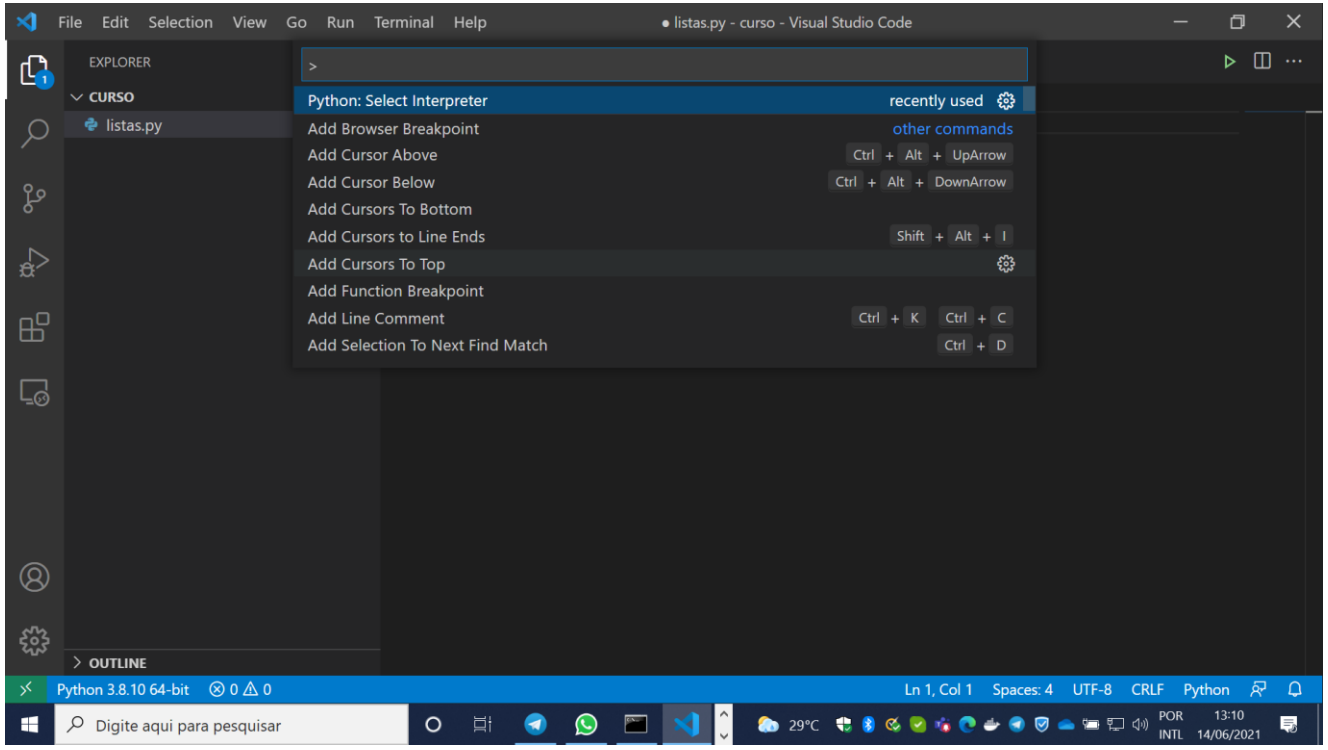
Listas

- Crie o seu primeiro arquivo python



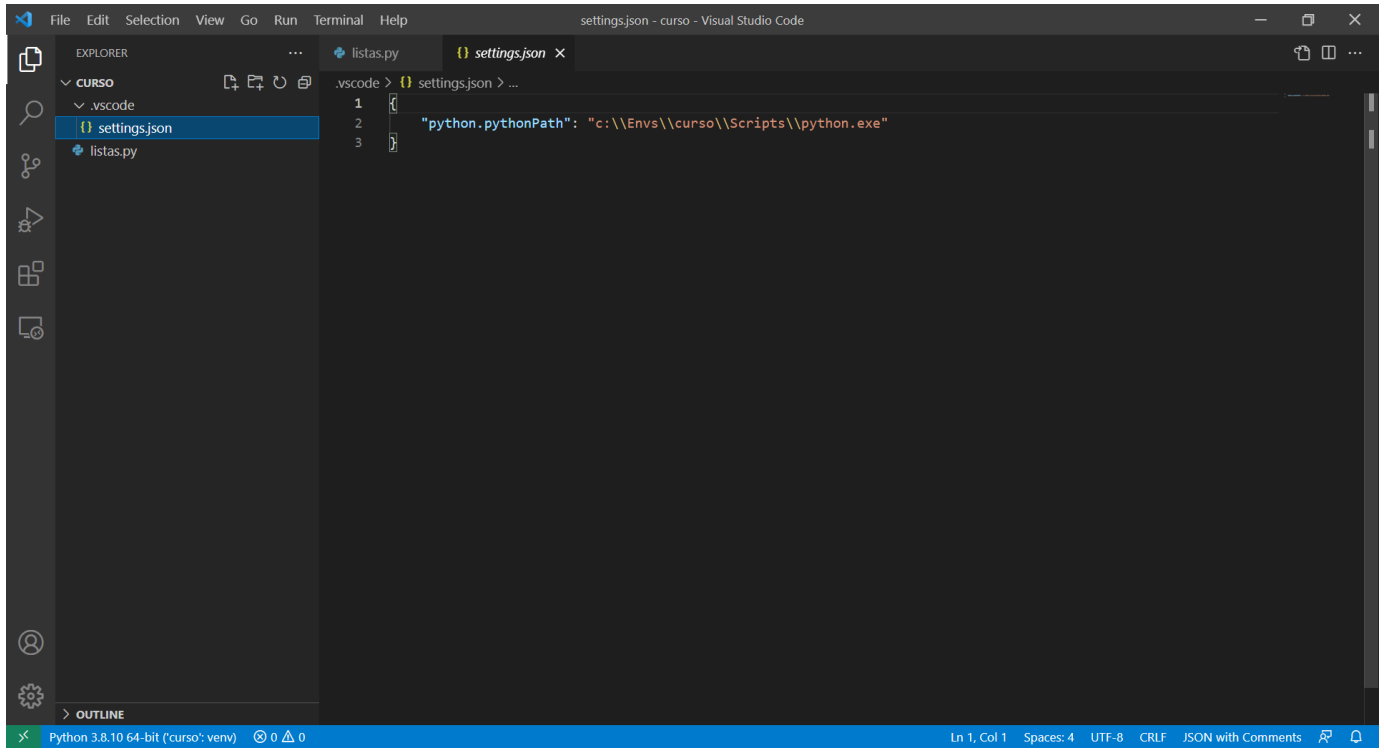
Listas

- Antes de dar prosseguimento é importante selecionar o interpretador do projeto. Nesse caso apontaremos para a pasta da **env** que criamos. Pressione **Ctrl + Shift + P**



Listas

- Pronto! O **VS Code** irá criar um arquivo **JSON** com a configuração do interpretador.



Listas

- Agora sim. Vamos falar das **listas**!!!
- **Listas** são mutáveis, ou seja, podemos alterar seus valores.

```
numeros = [10, 20, 30, 40, 50]

numeros.append(60)
numeros.append(70)

print(numeros)

print(f'Imprimindo a primeira posição da lista: {numeros[0]}')

# outras forma de declarar

nomes = list()

nomes.append('Osenias')
nomes.append('Izzie')
nomes.append('Rodrigo')

print(f'Lista de nomes {nomes}')
```

Listas

- Manipulação de **listas**

listas.py > ...

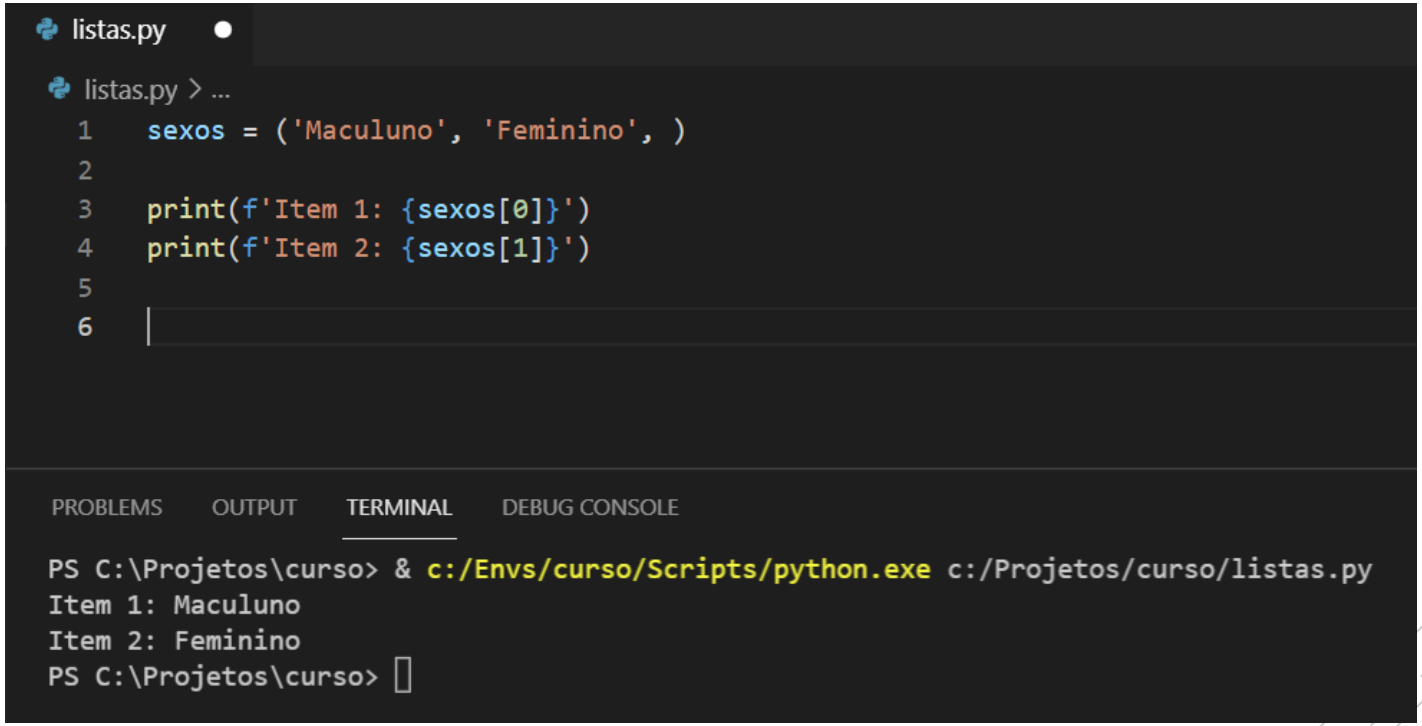
```
1   numeros = [10, 20, 30, 40, 50]
2
3   print(numeros[0:1])
4   print(numeros[:1])
5   print(numeros[0:2])
6   print(numeros[:2])
7   print(numeros[-1])
8   print(numeros[-2])
9   print(numeros[-3])
10
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/listas.py
[10]
[10]
[10, 20]
[10, 20]
50
40
30
PS C:\Projetos\curso>
```

Tuplas

- **Tuplas** também são listas, porém imutáveis, ou seja, uma vez declarada com seus devidos valores a mesma não pode ser alterada. Geralmente usamos tuplas para representar valores concretos, como por exemplo sexo de uma pessoa.



Dicionários

- **Dicionários** são tipos que nos permitem armazenar chave e valor.

dicionarios.py

```python
funcionario = {}

funcionario['nome'] = 'Osenias Oliveira'
funcionario['sexo'] = 'Masculino'
funcionario['idade'] = 32

print(funcionario)

outro_funcionario = dict()

outro_funcionario['nome'] = 'Izzie'
outro_funcionario['sexo'] = 'Feminino'
outro_funcionario['idade'] = 2

print(outro_funcionario)

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/dicionarios.py
{'nome': 'Osenias Oliveira', 'sexo': 'Masculino', 'idade': 32}
{'nome': 'Izzie', 'sexo': 'Feminino', 'idade': 2}
PS C:\Projetos\curso>
```

Estruturas de controle

estruturas_controle.py > ...

```python
idade = int(input('Digite sua idade: '))

if idade >= 18:
    print('Maior de idade')
elif idade >= 12 and idade < 18:
    print('Adolescente')
else:
    print('Criança')
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/estruturas_controle.py
Digite sua idade: 33
Maior de idade
PS C:\Projetos\curso>
```

Estruturas de Repetição

- **For**

estruturas_repeticao.py

```python
contador = 10

for c in range(contador):
    print(f'Contador: {c}')

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/estruturas_repeticao.py
Contador: 0
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
Contador: 6
Contador: 7
Contador: 8
Contador: 9
```

Estruturas de repetição

- **While**

estruturas_repeticao.py

```python
while True:
    valor = int(input('Informe um valor: '))
    if valor == 0:
        break

```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/estruturas_repeticao.py
Informe um valor: 10
Informe um valor: 20
Informe um valor: 0
PS C:\Projetos\curso>
```

Exercícios

- Dada uma lista de valores inteiros criar um programa para somar os valores pares e valores ímpares
- Dada uma palavra criar uma programa para retornar a quantidade de vogais e quantidade de consoantes
- Criar um programa para adicionar funcionários. Os funcionários devem ter nome, sexo e salário o programa deve retornar a soma de salários dos homens e soma dos salários das mulheres

Funções

funcoes.py > ...

```python
def somar(a, b):
    return a + b

def subtrair(a:int, b:int):
    return a - b

def multiplicar(a:int, b:int) -> float:
    return a * b


print(somar(20, 20))
print(subtrair(20, 20))
print(multiplicar(b=10, a=20))

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/funcoes.py
40
0
200
```

Funções

- Função anônima

funcoes.py > ...

```python
somar = lambda parametro_a, parametro_b : parametro_a + parametro_b

def somar_e_depois_multiplicar(a, b, multiplicador):
    return somar(parametro_a=a, parametro_b=b) * multiplicador

print(somar_e_depois_multiplicar(10, 20, 2))
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/funcoes.py
60
```

Funções

- Parâmetros de funções como lista

funcoes.py > ...

```python
def somar(*numeros):
    acumulador = 0
    for n in numeros:
        acumulador += n
    return acumulador


print(somar(10, 20, 30))

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/funcoes.py
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/funcoes.py
60
```

Funções

- Parâmetros de funções como dicionário

```python
def calcular(**kwargs):
    operacao = kwargs.get('operacao', '+')
    valor1 = kwargs.get('valor1', 0)
    valor2 = kwargs.get('valor2', 0 if not operacao == '/' else 1)
    return eval(f'{valor1} {operacao} {valor2}')


print(calcular(valor1=1, valor2=10, operacao='+'))

parametros = {
    'valor1': 1,
    'valor2': 10,
    'operacao': '+'
}

print(calcular(**parametros))
```

Funções parciais

funcoes.py > ...

```python
from functools import partial

def calcular_salario(valor_hora, horas_trabalhadas, horas_extras, extra_100 = False):
    if extra_100:
        return ( valor_hora * horas_trabalhadas ) + (valor_hora * horas_extras)
    else:
        return ( valor_hora * horas_trabalhadas ) + ((valor_hora / 2) * horas_extras)

valor_hora = float(input('Informe o valor da hora: '))
horas_trabalhadas = int(input('Informe as horas trabalhadas: '))
horas_extras = int(input('Informe as horas extras: '))

salario_base = partial(
    calcular_salario,
    valor_hora=valor_hora,
    horas_trabalhadas=horas_trabalhadas,
    horas_extras=5
)

extra_100 = salario_base(extra_100=True)
extra_50 = salario_base()

print(f'Salário com extra de 100%: {extra_100}' )
print(f'Salário com extra de 50%: {extra_50}')

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Projetos\curso> & c:/Envs/curso/Scripts/python.exe c:/Projetos/curso/funcoes.py
Informe o valor da hora: 2
Informe as horas trabalhadas: 20
Informe as horas extras: 1
Salário com extra de 100%: 50.0
Salário com extra de 50%: 45.0
PS C:\Projetos\curso>
```

Funções de string

funcoes_string.py > ...

```python
empresa = 'FPF Tech'

print('len')
print(len(empresa))

print('count')
print(empresa.count('F'))

print('find')
print(empresa.find('T'))

print('lower')
print(empresa.lower())
```

Funções de string

funcoes_string.py > ...

```python
empresa = 'FPF Tech'

print('upper')
print(empresa.upper())

print('split')
print(empresa.split(' '))

print('capitalize')
print(empresa.lower().capitalize())

print('replace')
print(empresa.replace('Tech', 'Fundação'))
```

Funções de string

funcoes_string.py > ...

```python
empresa = 'FPF Tech'
nomes = ['Osenias', 'Izzie']

print('title')
print(empresa.lower().title())

print('join')
print(', '.join(nomes))

```

Funções built-in

funcoes_string.py

```
1   print(float('10'))
2
3   print(int('10'))
4
5   print(str(10))
6
7   print(bool(1))
8
9   print('Hello World')
10
11  print(len([1, 2, 3, 4, 5, 6]))
12
13  print(max([1, 2, 3, 4, 5, 6]))
14
15  print(min([1, 2, 3, 4, 5, 6]))
```

Funções built-in

funcoes_string.py > ...

```python
for index, element in enumerate([1, 2, 3, 4, 5]):
    print(f'index: {index}')
    print(f'element: {element}')

print(sum([1, 2, 3, 4, 5]))

input('Digite seu nome: ')
```

Funções built-in

```python
frutas = ['maçã', 'laranja', 'pessego', 'morango']

def tamanho_item(item: str):
    return len(item)

transformado = map(tamanho_item, frutas)

transformado_2 = map(lambda item: len(item), frutas)

print(f'transformado: {list(transformado)}')
print(f'transformado 2: {list(transformado_2)}')
```

Funções built-in

```python
numeros = [100, 200, 300, 1, 4, 2, 3, 5, 6]

lista_ordenada = sorted(numeros)
lista_ordenada_desc = sorted(numeros, reverse=True)

print(lista_ordenada)
print(lista_ordenada_desc)
```

Funções built-in

```
numeros = [100, 200, 300, 1, 4, 2, 3, 5, 6]

def par(item):
    return item % 2 == 0


pares = filter(par, numeros)
pares_2 = filter(lambda item: item % 2 == 0, numeros)

print(list(pares))
print(list(pares_2))
```

Funções built-in

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Compreensão de listas

- Dada uma lista de valores inteiros criar um algoritmo para a partir dessa lista criar uma lista com os valores menores que 100

compreensao.py > ...

```python
numeros = [1, 2, 3, 4, 5, 100, 200, 500, 600]
menores_que_100 = []

for n in numeros:
    if n < 100:
        menores_que_100.append(n)

print(menores_que_100)
```

Compreensão de listas

- Dada uma lista de valores inteiros criar um algoritmo para a partir dessa lista criar uma lista com os valores menores que 100

compreensao.py > ...

```
1   numeros = [1, 2, 3, 4, 5, 100, 200, 500, 600]
2   menores_que_100 = [n for n in numeros if n < 100]
3   print(menores_que_100)
```

Compreensão de listas

compreensao.py > ...

```python
funcionarios = []

f1 = {'nome': 'Osenias', 'salario': 2000, 'sexo': 'M'}
f2 = {'nome': 'Izzie', 'salario': 5000, 'sexo': 'F'}
f3 = {'nome': 'Taty', 'salario': 10000, 'sexo': 'F'}
f4 = {'nome': 'Rodrigo', 'salario': 2500, 'sexo': 'M'}

funcionarios.append(f1)
funcionarios.append(f2)
funcionarios.append(f3)
funcionarios.append(f4)

masculino = sum([f['salario'] for f in funcionarios if f['sexo'] == 'M'])
feminino = sum([f['salario'] for f in funcionarios if f['sexo'] == 'F'])

print(masculino)
print(feminino)
```

Compreensão de listas

compreensao.py > ...

```python
from collections import namedtuple
from datetime import date

Funcionario = namedtuple('Funcionario', ['nome', 'salario', 'sexo', 'aniversario'])

f1 = Funcionario(nome='Osenias', salario=2500, sexo='M', aniversario=date(1987, 1, 1))
f2 = Funcionario(nome='Tatiany', salario=3500, sexo='F', aniversario=date(2000, 1, 1))
f3 = Funcionario(nome='Izzie', salario=5500, sexo='F', aniversario=date(2005, 1, 1))
f4 = Funcionario(nome='Rodrigo', salario=7500, sexo='M', aniversario=date(2006, 1, 1))

funcionarios = []
funcionarios.append(f1)
funcionarios.append(f2)
funcionarios.append(f3)
funcionarios.append(f4)

print(funcionarios)

print(min([f.aniversario for f in funcionarios]))
```

Iterators

- Qualquer coisa que podemos fazer um **for** pode ser considerado um **iterable**.
  - Listas
  - Tuplas
  - Strings
  - Dicionários
- A ideia principal de **iterators** é resolver problemas de sequências grandes, que provavelmente não cabem na memória.
- Nos tipos que iteráveis que já conhecemos é possível fazer o **slice** pois sabemos que o valor sempre está disponível;

Iterators

- Exemplos de **iterators**
  - **map**
  - **filter**
  - **reverserd**

Iterators

- Para conseguirmos acessar os itens de um **iterator** é preciso que façamos o evaluate dos itens do **iterator** ou utilizar o método **next**.

```
numeros = iter([1, 2, 3])

print(list(numeros))

numeros = iter([1, 2, 3])

print(next(numeros))
print(next(numeros))
print(next(numeros))
```

Iterators

- Para deixar um pouco mais claro vamos criar uma função que funciona como um gerador de **iterators**.

```
def contador():
    indice = 0
    while True:
        yield indice
        indice += 1


gerador = contador()

print(next(gerador))
print(next(gerador))
print(next(gerador))
print(next(gerador))
print(next(gerador))
print(next(gerador))
```

Iterators

- Perceba que cada vez que chamamos a função **next** para a função geradora de um **iterator** acionamos um novo **loop** no nosso **while True** isso nos permite navegar item a item.

```
PS C:\Projetos\curso_python> & c:/Envs/curso_python/Scripts/python.exe
radores.py
0
1
2
3
4
5
```

Iterators

- Calma! Que diabos é esse **yeild?** 😐
- O **yield** nos ajuda a criar as funções geradoras, pois ele nos retorna o valor e mantém o estado de onde parou na última execução. Nos permitindo percorrer uma lista sob demanda.

Iterators

- Vamos fazer mais um exemplo para assimilar o conceito e como funcionam os **iterators.**

```
tabuada_eager = [f'{n} * 1 = { n * 1 }' for n in range(1, 11)]
print('tabuada_eager'.upper())
print(tabuada_eager)
print('*' * 100)
print('\n'.join(tabuada_eager))

tabuada_lazy = (f'{n} * 1 = { n * 1 }' for n in range(1, 11))
print('tabuada_lazy'.upper())
print(tabuada_lazy)

print(next(tabuada_lazy))
print(next(tabuada_lazy))
```

Iterators

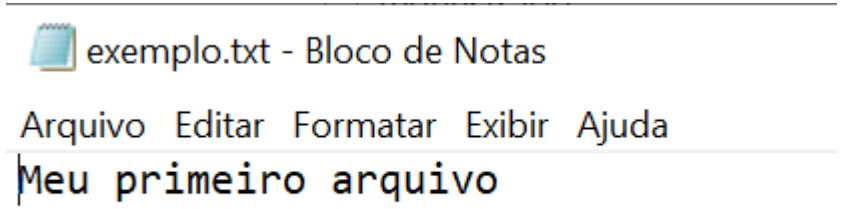
- Vamos fazer mais um exemplo para assimilar o conceito e como funcionam os **iterators.**

```
TABUADA_EAGER
['1 * 1 = 1', '2 * 1 = 2', '3 * 1 = 3', '4 * 1 = 4', '5 * 1 = 5', '6 * 1 = 6', '7 * 1 = 7', '8 * 1 = 8', '9 * 1 = 9', '
10 * 1 = 10']
**********************************************************************************************
1 * 1 = 1
2 * 1 = 2
3 * 1 = 3
4 * 1 = 4
5 * 1 = 5
6 * 1 = 6
7 * 1 = 7
8 * 1 = 8
9 * 1 = 9
10 * 1 = 10
TABUADA_LAZY
<generator object <genexpr> at 0x0000025C8535FF90>
1 * 1 = 1
2 * 1 = 2
```

Arquivos

- Escrevendo em um arquivo

```
arquivo = open('c:\\tmp\\exemplo.txt', 'w+')

arquivo.write('Meu primeiro arquivo')

arquivo.close()
```



Arquivos

- Lendo conteúdo de um arquivo

```
arquivo = open('c:\\tmp\\exemplo.txt', 'r')

conteudo = arquivo.read()

print(conteudo)

arquivo.close()
```

Arquivos

- Adicionando novas linhas no arquivo

```
arquivo = open('c:\\tmp\\exemplo.txt', 'a')

conteudo = arquivo.write('\nSegunda linha do arquivo')

print(conteudo)

arquivo.close()
```

Arquivos

- Adicionando novas linhas no arquivo

```
arquivo = open('c:\\tmp\\exemplo.txt', 'r')

linhas = arquivo.readlines()

for l in linhas:
    print(l)

arquivo.close()
```

Arquivos

- Utilizando bloco **with**

```
with open('c:\\tmp\\exemplo.txt', 'r') as arquivo:
    linhas = arquivo.readlines()
    for l in linhas:
        print(l)
```

Classes, atributos e métodos

- Declaração

```
class Funcionario:
    pass

class Pessoa(object):
    pass
```

Classes, atributos e métodos

- Atributos estáticos e da instância

```
class Funcionario:
    nome = None
    idade = 0

class Pessoa(object):
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
```

Classes, atributos e métodos

```
class Pessoa(object):
    def __init__(self, nome = None, idade = None):
        self.nome = nome
        self.idade = idade
```

Classes, atributos e métodos

```
p1 = Pessoa('Osenias', 33)
p2 = Pessoa()

p2.nome = 'Izzie'
p2.idade = 2

print(p1)
print(p2)
```

Classes, atributos e métodos

```python
from datetime import datetime, date


class Pessoa(object):
    def __init__(self, nome = None, aniversario = None):
        self.nome = nome
        self.aniversario = aniversario


    def __str__(self) -> str:
        return self.nome

    @property
    def idade(self):
        hoje = datetime.now().date()
        diferenca = hoje - self.aniversario
        return int((diferenca).days / 365)
```

Classes, atributos e métodos

```
p1 = Pessoa()
p1.nome = 'Osenias'
p1.aniversario = date(1987, 10, 28)
print(p1.idade)
```

Herança

```python
class Pessoa(object):
    def __init__(self, nome = None):
        self.nome = nome


    def __str__(self) -> str:
        return self.nome


class PessoaJuridida(Pessoa):
    def __init__(self, nome = None, cnpj = None):
        super().__init__(nome=nome)
        self.cnpj = cnpj


class PessoaFisica(Pessoa):
    def __init__(self, nome = None, cpf = None):
        super().__init__(nome=nome)
        self.cpf = cpf
```

Herança

```
fpf = PessoaJuridida()
fpf.nome = "FPF Tech"
fpf.cnpj = "000.000.000/0001-20"

print(fpf)

osenias = PessoaFisica()
osenias.nome = "Osenias Oliveira"
osenias.cpf = "000.000.000-00"

print(osenias)
```

Herança

- Sobrecarga de métodos.
- Mas... não recomendo ☺

```python
from functools import singledispatchmethod

class Calculadora:

    @singledispatchmethod
    def soma(self, a, b):
        raise NotImplementedError('aqui não vai ser implmentado')

    @soma.register
    def inteiros(self, a: int, b: int):
        print('int')
        return a + b

    @soma.register
    def strings(self, a: str, b: str):
        print('str')
        return a + b



c = Calculadora()
print(c.soma(10, 10))
print(c.soma('10', '10'))
```

Herança

- Herança múltipla. Isso aqui funciona de verdade.

```python
class Jogador:
    def bater_penalti(self):
        print('Eu sei bater penalti')


class Nadador:
    def nadar(self):
        print('Eu sei nadar')


class Pessoa(Jogador, Nadador):
    def __init__(self, nome = None, peso = None):
        self.nome = nome
        self.peso = peso



p = Pessoa()
p.nome = "Osenias"
p.peso = 80
p.bater_penalti()
p.nadar()
```

Módulos

- Em python um arquivo com extensão **py** é um módulo então dentro de um módulo eu posso ter N classes, atributos e métodos.
- Podemos ter um módulo só com métodos utils por exemplo.


```python
def formatar_data():
    raise NotImplementedError("Aqui será formatada a data")

def add_mascarara(campo: str):
    raise NotImplementedError("Aqui será adicoonada a máscara")

def converter_para_decimal():
    raise NotImplementedError("Aqui será convertido para decimal")
```

Módulos

- Podemos também ter um módulo só com variáveis de configuração por exemplo.

settings.py

```
PATH_API = "http://127.0.0.1:9000/curso"
DEFAULT_USER_API = "admin"
DEFAULT_PASSWORD_API = "admin"

TIME_ZONE = "America/Manaus"
```

Módulos

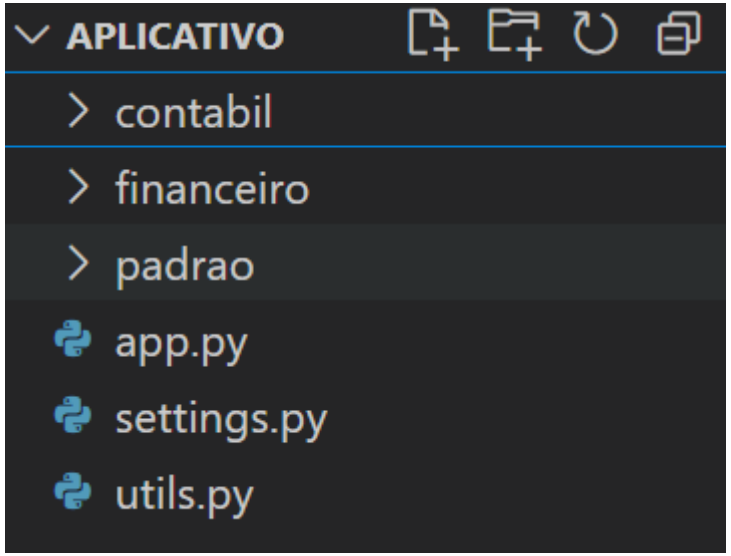
- Podemos também ter um módulo apenas com classes.

models.py

```python
class Pessoa:
    def __init__(self, nome = None, sexo = None):
        self.nome = nome
        self.sexo = sexo


class Usuario:
    def __init__(self, login = None, senha = None):
        self.login = login
        self.senha = senha
```

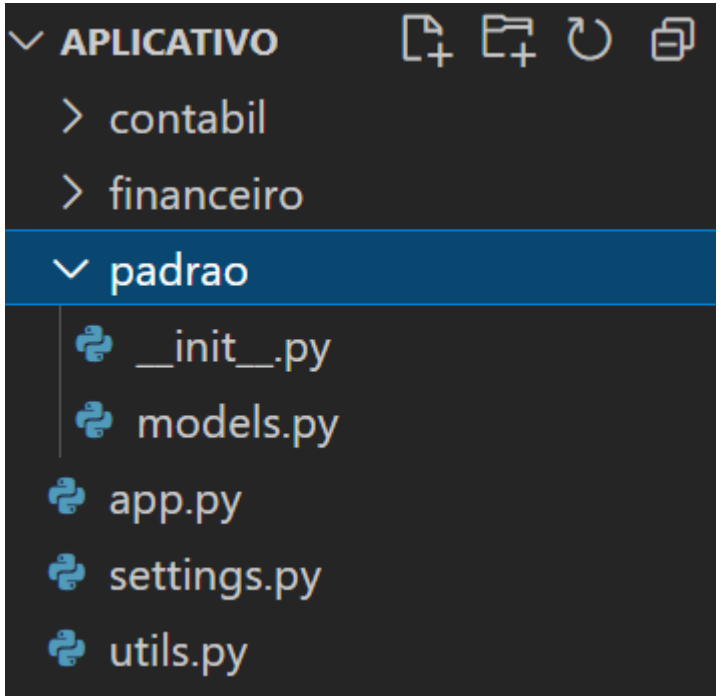
Módulos

- Também podemos criar estrutura de pastas em nosso projeto com a ideia de organizar melhor os contextos.



Módulos

- Ao criarmos pastas na estrutura do projeto para que as mesmas possam ser importadas quando usadas, representando assim uma estrutura similar a um pacote devemos sempre adicionar um arquivo **__init__.py,** o mesmo pode está com o conteúdo vazio, mas é necessário para inicialização da pasta como um módulo.



Importações

- Podemos fazer a importação de várias maneiras diferentes.
  - Importando pelo nome do módulo.
  - Importando uma função específica que será usada.
  - Importando uma classe específica que será usada.
- Você pode também dar uma apelido para a importação e no contexto daquele módulo python utilizar o apelido para fazer as chamadas.

Importações

```
from padrao import models


p = models.Pessoa()
p.nome = "Osenias"
p.sexo = "Masculino"

print(p.nome)
```

```
from utils import add_mascarara

print(add_mascarara('campo'))
```

Importações

```python
from padrao.models import Pessoa


p = Pessoa()
p.nome = "Osenias"
p.sexo = "Masculino"

print(p.nome)
```

```python
from padrao.models import Pessoa as MinhaPessoa


p = MinhaPessoa()
p.nome = "Osenias"
p.sexo = "Masculino"

print(p.nome)
```

Exceções

- No python o tratamento de exceção é muito similar a outras linguagens, você tem um bloco onde o código tentará ser executado e outro para caso ocorra algum erro.

```
try:
    numero_digitado = int(input('Digite um valor inteiro: '))
except ValueError:
    print('Erro. Você deve digitar um valor inteiro.')
```

Exceções

- Podemos exibir o erro rastreado.

```
try:
    numero_digitado = int(input('Digite um valor inteiro: '))
except ValueError as erro:
    print(f'Erro. Você deve digitar um valor inteiro.\n{erro}')
```

Exceções

- Podemos lançar a exceção para que seja tratada em outro momento.

```python
class Calculadora:
    def divisao(self, a, b):
        try:
            return a / b
        except ZeroDivisionError:
            raise Exception('Não posso dividir por zero :(')


try:
    c = Calculadora()
    c.divisao(10, 0)
except Exception as erro:
    print(erro)

```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Projetos\aplicativo> & C:/Users/osenias.oliveira/AppData/Local/P
38/python.exe c:/Projetos/aplicativo/app.py
Não posso dividir por zero :(
PS C:\Projetos\aplicativo>
```

Exceções

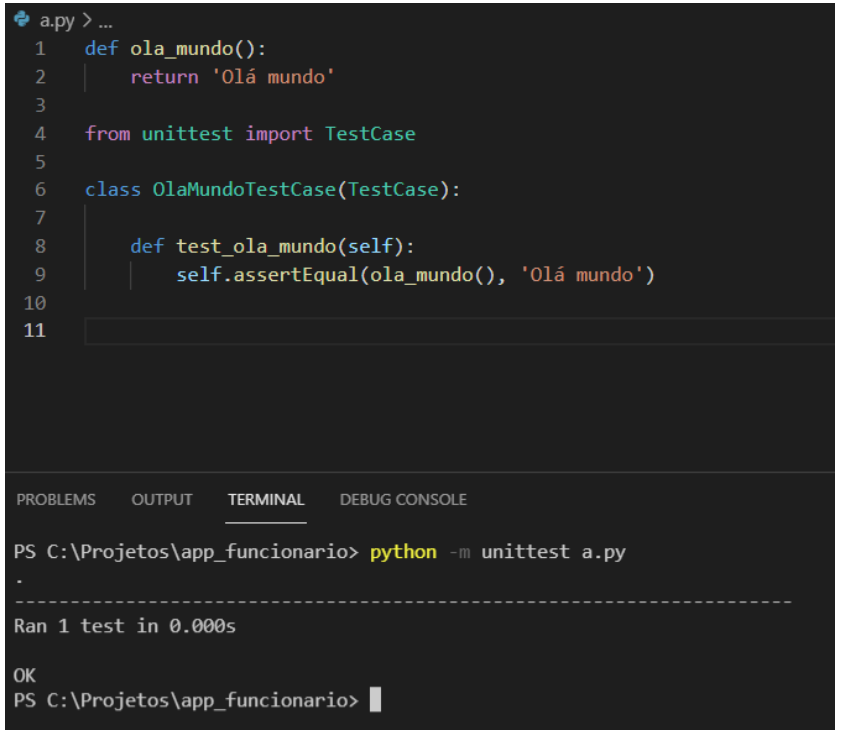
- Podemos também criar uma classe de exceção customizada.

```python
class ExcecaoCustomizada(Exception):
    def __init__(self, mensagem, mais_um_valor) -> None:
        self.mensagem = mensagem
        self.mais_um_valor = mais_um_valor


    def __str__(self) -> str:
        return f'Erro customizado.\n{self.mensagem}\n{self.mais_um_valor}'


class Calculadora:
    def divisao(self, a, b):
        try:
            return a / b
        except ZeroDivisionError:
            raise ExcecaoCustomizada('Não posso dividir por zero :(', 'É um erro mesmo')


try:
    c = Calculadora()
    c.divisao(10, 0)
except Exception as erro:
    print(erro)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

licativo/app.py
Erro customizado.
Não posso dividir por zero :(
É um erro mesmo
```

Testes unitários

- É importante garantirmos a qualidade do que estamos entregando. Para isso é necessário que façamos vários tipos de testes, testes exploratórios, testes de performance, testes unitários e outros.
- Vamos ver um pouco sobre como podemos testar nossas classes e métodos com python.



Testes unitários

```python
class Calculadora:
    def __init__(self, a, b) -> None:
        self.a = a
        self.b = b


    def somar(self):
        return self.a + self.b


    def subtrair(self):
        return self.a - self.b


    def multiplicar(self):
        return self.a * self.b


    def dividir(self):
        try:
            return self.a / self.b
        except ZeroDivisionError:
            raise ZeroDivisionError('Divisão por zero não permitida')
```

Testes unitários

```python
from unittest import TestCase
from calculadora import Calculadora

class CalculadoraTestCase(TestCase):

    def setUp(self) -> None:
        self.a = 50
        self.b = 50


    def test_somar(self):
        resultado = Calculadora(self.a, self.b).somar()
        self.assertEqual(resultado, 100)


    def test_subtrair(self):
        resultado = Calculadora(self.a, self.b).subtrair()
        self.assertEqual(resultado, 0)


    def test_multiplicar(self):
        resultado = Calculadora(self.a, self.b).multiplicar()
        self.assertEqual(resultado, 2500)
```

Testes unitários

```python
def test_dividir(self):
    resultado = Calculadora(self.a, self.b).dividir()
    self.assertEqual(resultado, 1)


def test_dividir_com_erro(self):
    self.b = 0
    self.assertRaises(ZeroDivisionError, lambda: Calculadora(self.a, self.b).dividir())


def tearDown(self) -> None:
    self.b = 50
```

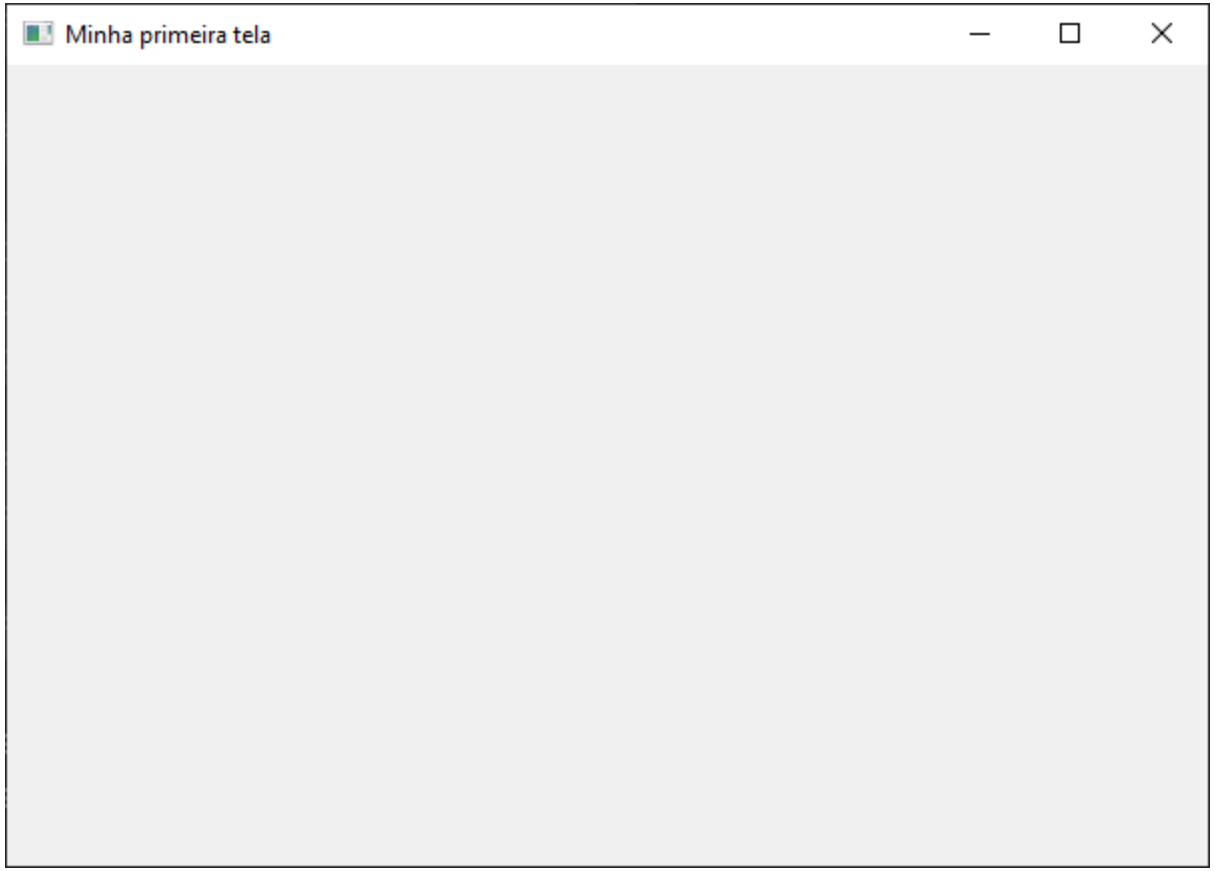
Outros pacotes

- Vamos conhecer um pouco da biblioteca para trabalhar com aplicações desktop.
- **PyQT5** é um empacotador do **QT** para a linguagem python.
- Suporta as plataformas **UNIX, Linux, Windows e Mac OS**
- Para começarmos execute o comando **pip install pyqt5** dentro da sua virtual env.

Outros pacotes

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow


class MainForm(QMainWindow):

    def __init__(self, parent = None):
        super().__init__(parent=parent)

        self.setWindowTitle('Minha primeira tela')
        self.setGeometry(20, 20, 800, 400)



app = QApplication(sys.argv)
main = MainForm()
main.show()
sys.exit(app.exec_())
```

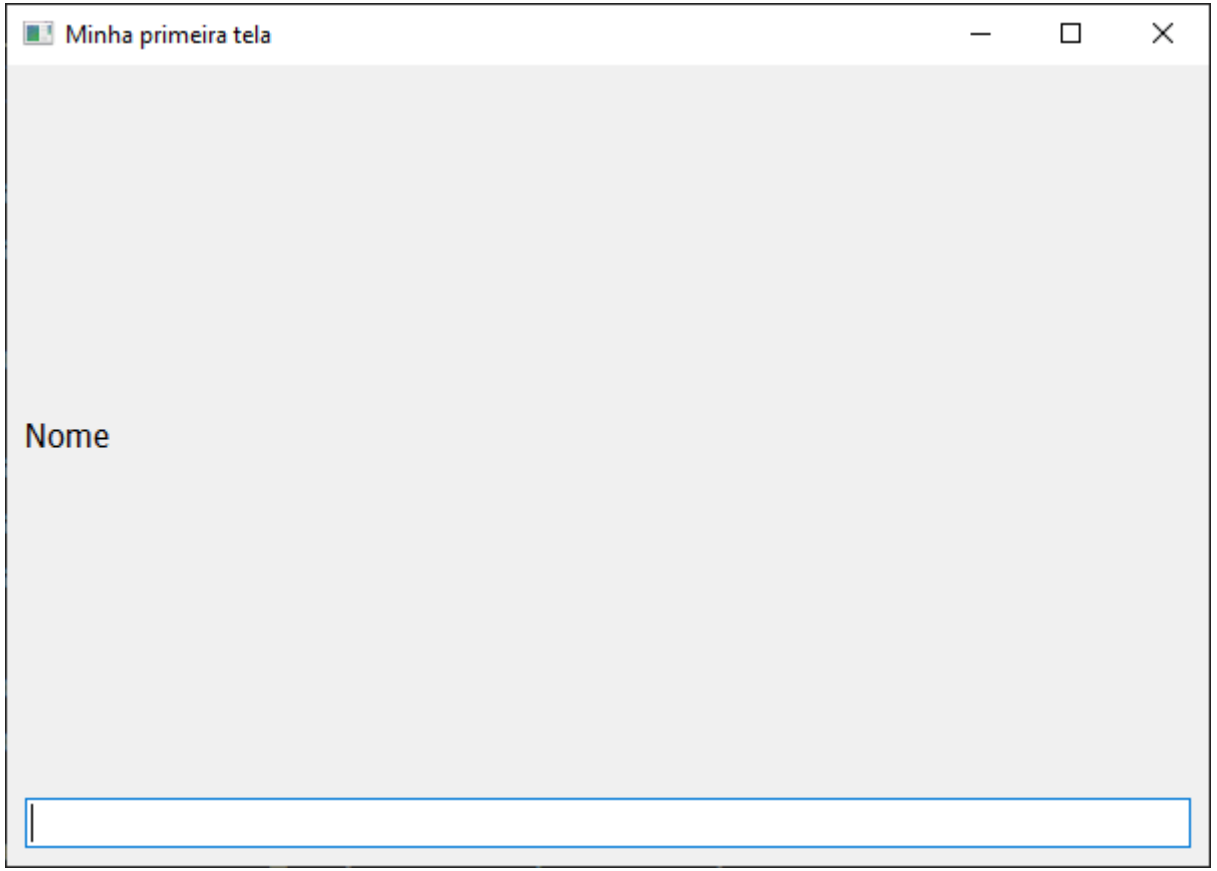
Outros pacotes



Outros pacotes

```
import sys
from PyQt5.QtWidgets import QApplication, QLineEdit, QMainWindow, QVBoxLayout, QLabel, QWidget


class MainForm(QMainWindow):

    def __init__(self, parent = None):
        super().__init__(parent=parent)

        self.label_nome = QLabel()
        self.label_nome.setText('Nome')

        self.edt_nome = QLineEdit()

        vertical_layout = QVBoxLayout()
        vertical_layout.addWidget(self.label_nome)
        vertical_layout.addWidget(self.edt_nome)

        self.componentes = QWidget()
        self.componentes.setLayout(vertical_layout)

        self.setCentralWidget(self.componentes)

        self.setWindowTitle('Minha primeira tela')
        self.setGeometry(10, 10, 600, 400)
```

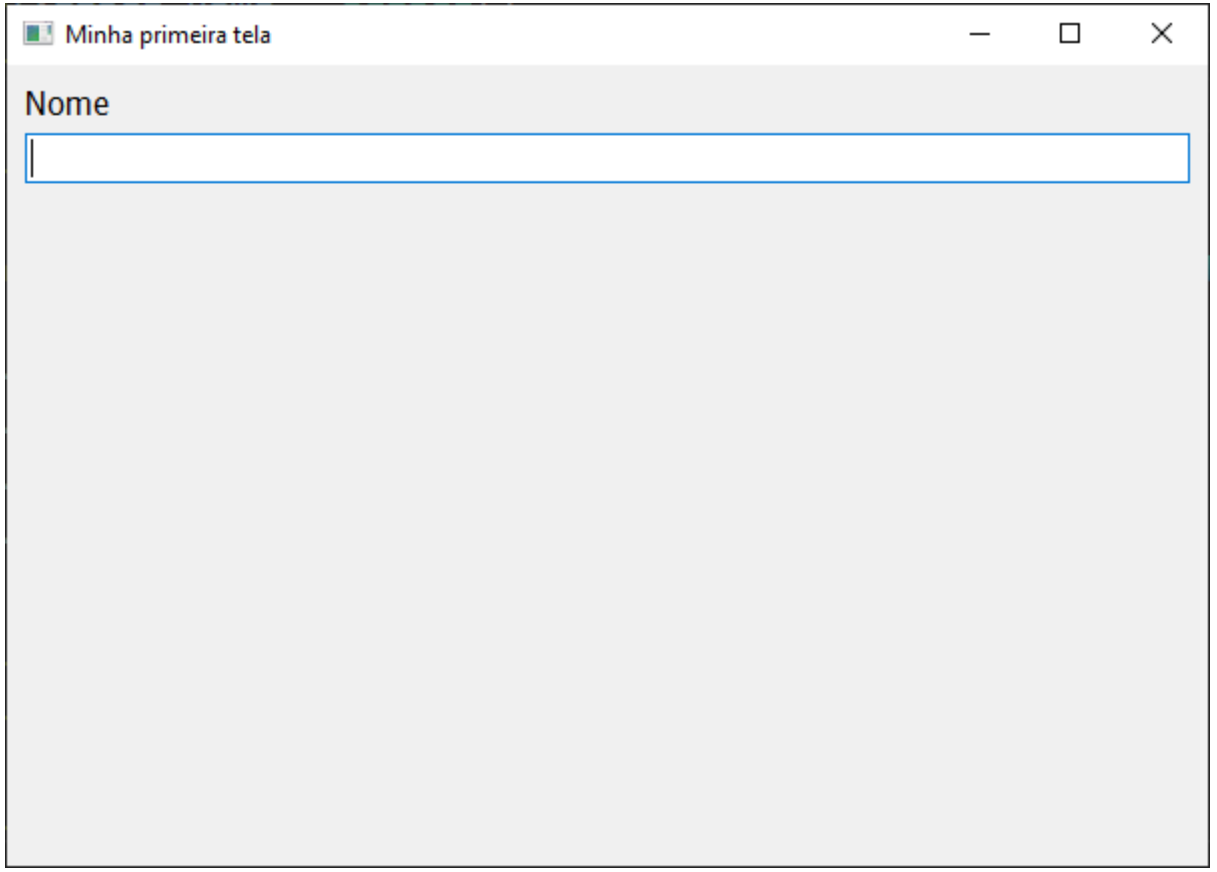
Outros pacotes



Outros pacotes

```python
import sys
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QLineEdit, QMainWindow, QSpacerItem, QVBoxLayout, QLabel, QWidget


class MainForm(QMainWindow):

    def __init__(self, parent = None):
        super().__init__(parent=parent)

        self.label_nome = QLabel()
        self.label_nome.setText('Nome')

        self.edt_nome = QLineEdit()

        space = QSpacerItem(0, 0, QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Expanding)

        vertical_layout = QVBoxLayout()
        vertical_layout.addWidget(self.label_nome)
        vertical_layout.addWidget(self.edt_nome)
        vertical_layout.addItem(space)

        self.componentes = QWidget()
        self.componentes.setLayout(vertical_layout)

        self.setCentralWidget(self.componentes)

        self.setWindowTitle('Minha primeira tela')
        self.setGeometry(10, 10, 600, 400)
```

Outros pacotes



Outros pacotes

```
import sys
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QLineEdit, QMainWindow, QPushButton, QSpacerItem, QVBoxLayout, QLabel, QWidget


class MainForm(QMainWindow):

    def __init__(self, parent = None):
        super().__init__(parent=parent)

        self.label_nome = QLabel()
        self.label_nome.setText('Nome')

        self.edt_nome = QLineEdit()

        self.button_adicionar = QPushButton('Adicionar')

        space = QSpacerItem(0, 0, QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Expanding)

        vertical_layout = QVBoxLayout()
        vertical_layout.addWidget(self.label_nome)
        vertical_layout.addWidget(self.edt_nome)
        vertical_layout.addWidget(self.button_adicionar)
        vertical_layout.addItem(space)

        self.componentes = QWidget()
        self.componentes.setLayout(vertical_layout)

        self.setCentralWidget(self.componentes)

        self.setWindowTitle('Minha primeira tela')
        self.setGeometry(10, 10, 600, 400)
```

Outros pacotes

- Evento de clique no botão.

```
self.button_adicionar = QPushButton('Adicionar')
self.button_adicionar.clicked.connect(self.clicando)
```

```
def clicando(self):
    print('cliquei no botão')
```

Outros pacotes

```
self.table = QTableWidget()
self.table.setColumnCount(2)
self.table.setHorizontalHeaderLabels(['Nome', 'Sexo'])
self.table.setRowCount(0)
```

```
vertical_layout.addWidget(self.table)
```

Outros pacotes

```
def popular_tabela(self):
    self.table.setRowCount(len(self.FUNCIONATIOS))
    for linha, valor in enumerate(self.FUNCIONATIOS):
        nome = QTableWidgetItem()
        nome.setText(valor['nome'])
        nome.setData(QtCore.Qt.UserRole, valor)

        sexo = QTableWidgetItem()
        sexo.setText(valor['sexo'])

        self.table.setItem(linha, 0, nome)
        self.table.setItem(linha, 1, sexo)
```

```
def __init__(self, parent = None):
    super().__init__(parent=parent)

    self.popular_tabela()
```

Outros pacotes